Pannon Egyetem
Műszaki Informatikai Kar
Informatikai Tudományok Doktori Iskola

Üzleti folyamatok online nyomon követése és javítása folyamatbányászati és mesterséges intelligencia algoritmusok alkalmazásával

Doktori (PhD) értekezés

Nagy Zsuzsanna

DOI:10.18136/PE.2025.939

Témavezető: Starkné dr. Werner Ágnes

2025

ÜZLETI FOLYAMATOK ONLINE NYOMON KÖVETÉSE ÉS JAVÍTÁSA FOLYAMATBÁNYÁSZATI ÉS MESTERSÉGES INTELLIGENCIA ALGORITMUSOK ALKALMAZÁSÁVAL

Az értekezés doktori (PhD) fokozat elnyerése érdekében készült a Pannon Egyetem Informatikai Tudományok Doktori Iskolája keretében

informatikai tudományok tudományágban

Írta: Nagy Zsuzsanna

Témavezető/i: Starkné dr. Werner Ágnes

Elfogadásra javaslom (igen / nem)

……..……
Starkné dr. Werner Ágnes

Az értekezést bírálóként elfogadásra javaslom:

Bíráló neve: …..….... igen /nem

…….
bíráló

Bíráló neve: …..….... igen /nem

…….
bíráló

A jelölt az értekezés nyilvános vitáján ….....%-ot ért el.

Veszprém,

…….…….
a Bíráló Bizottság elnöke

A doktori (PhD) oklevél minősítése….....
Veszprém,

……..…….
az EDHT elnöke

Tartalmi kivonat

A doktori értekezésemben folyamatbányászati és mesterséges intelligencia algoritmusok felhasználásával új megoldásokat dolgoztam ki az üzleti folyamatok online nyomon követéséhez és javításához.

A dolgozatom első fele a rövid átfutási idejű folyamatokra összpontosít, mint amilyenek például a gyártási folyamatok. A munkám egyik legfontosabb hozzájárulása az első több perspektívás online megfelelőség-ellenőrzési (MOCC) megoldás kifejlesztése. A valós folyamatokon végzett tesztek bizonyították az MOCC hatékonyságát mind több perspektívás, mind online megfelelőség-ellenőrzési módszerként. A folyamatadatok megfelelő időbeli megjelenítési módszereinek hiánya miatt új valós idejű vizualizációs technikákat is kidolgoztam. Ezek lehetővé teszik az eredeti eseményadatok és az igazítás-alapú megfelelőség-ellenőrzés kimeneteinek megjelenítését, miközben megfelelően kezelik az átfedő objektumokat. Egy valós folyamaton végzett vizsgálatok azt mutatták, hogy ezek a vizualizációs módszerek megkönnyítik az olyan anomáliák és azok kiváltó okainak felderítését, amelyek egyébként rejtve maradnának.

A dolgozatom második fele a szállítási folyamatokra összpontosít, és a kapacitáskorlátos ív útvonaltervezési problémával (CARP) és annak dinamikus változatával (DCARP) foglalkozik. A CARP számára létrehoztam az első közepes lépésméretű mozgási műveletet, a részútvonalterv műveletet, amelyet az útvonaltervek hatékonyabb optimalizálására fejlesztettem ki. A vizsgálatok azt mutatták, hogy átlagban jobb szolgáltatási tervet találva felülmúlja a meglévő műveleteket. A munkám másik jelentős hozzájárulása az első mesterséges méhcsalád (ABC) algoritmus kifejlesztése a CARP megoldására. Két változatot javasoltam: egy felfedezésre összpontosító változatot a CARP-hoz és egy feltárásra összpontosító változatot a DCARP-hoz. A DCARP megoldók teljesítményének valósághű szimulációs környezetben történő értékeléséhez egy olyan adat-vezérelt DCARP keretrendszert fejlesztettem ki, amely képes az összes lehetséges dinamikus eseményt kezelni. Ez a keretrendszer frissíti a probléma példányt és a szolgáltatási tervet az esemény létrehozó modul vagy egy forgalomszimulációs szoftver eseményeinek feldolgozásával és szükség esetén az újratervező moduljának meghívásával. A keretrendszer részeként javasoltam egy minimális újratervező (RR1) algoritmust, amely a szolgáltatási terv minimális módosításával biztosítja annak megvalósíthatóságát. A valós problémákon alapuló CARP-példákon végzett vizsgálatok azt mutatták, hogy a feltárásra összpontosító CARP-ABC algoritmus rövid időn belül közel optimális szolgáltatási terveket tud nyújtani. Az adat-vezérelt DCARP keretrendszerrel DCARP példákon végzett vizsgálatok azt mutatták, hogy az esetek többségében az RR1 algoritmus és a felfedezésre összpontosító CARP-ABC algoritmus együttes alkalmazása biztosítja a legjobb szolgáltatási terveket.

Abstract

In my PhD thesis, I developed novel solutions for the online monitoring and improvement of business processes by incorporating process mining and artificial intelligence algorithms.

The first half of my thesis focuses on processes with short lead times, such as manufacturing processes. A key contribution of my work is the development of the first multi-perspective online conformance checking (MOCC) solution. Tests on real-life processes demonstrated the effectiveness of the MOCC as both a multi-perspective and an online conformance checking method. To address the lack of proper temporal visualization methods for process data, I also developed new real-time visualization techniques. These allow displaying the original event data and the outputs of alignment-based conformance checking, while correctly handling overlapping objects. Tests on a real-life process showed that these visualization methods facilitate the detection of anomalies and their root causes, which would otherwise remain hidden.

The second half of my thesis focuses on transportation processes and deals with the capacitated arc routing problem (CARP) and its dynamic variant (DCARP). I introduced the first medium step-size move operator for the CARP, the sub-route plan operator, designed to optimize route plans more effectively. Tests showed that, on average, it outperforms existing operators in finding better service plans. Another significant contribution of my work is the development of the first Artificial Bee Colony (ABC) algorithm for solving the CARP. I proposed two versions: an exploration-focused version for the CARP and a discovery-focused version for the DCARP. To evaluate the performance of DCARP solvers in realistic simulated environments, I developed a data-driven DCARP framework capable of handling all possible dynamic events. This framework updates the problem instance and the service plan by processing events from the event generation module or a traffic simulation software and invoking its re-scheduling module when necessary. As part of the framework, I proposed a minimal rerouting (RR1) algorithm, which ensures feasibility with minimal modifications to the service plan. Experiments on CARP instances based on real-life problems showed that the exploration-focused CARP-ABC algorithm can provide near-optimal service plans within a short timeframe. Experiments performed on DCARP instances using the data-driven DCARP framework showed that the combination of the RR1 algorithm and the discovery-focused CARP-ABC algorithm provides the best service plans in most of the cases.

Resumen

En mi tesis doctoral, desarrollé soluciones innovadoras para el monitoreo y mejora en línea de los procesos de negocio mediante la incorporación de minería de procesos y algoritmos de inteligencia artificial.

La primera mitad de mi tesis se centra en procesos con plazos de ejecución cortos, como los procesos de fabricación. Una de las principales contribuciones de mi trabajo es el desarrollo de la primera solución de comprobación de conformidad en línea multi-perspectiva (MOCC). Las pruebas realizadas en procesos reales demostraron la efectividad del MOCC tanto como método de comprobación de conformidad multi-perspectiva como en línea. Para abordar la falta de métodos adecuados de visualización temporal para los datos de procesos, también desarrollé nuevas técnicas de visualización en tiempo real. Estas permiten mostrar los datos originales de los eventos y los resultados de la comprobación de conformidad basada en alineación, al tiempo que gestionan correctamente los objetos superpuestos. Las pruebas realizadas en un proceso real demostraron que estos métodos de visualización facilitan la detección de anomalías y sus causas raíz, las cuales de otro modo quedarían ocultas.

La segunda mitad de mi tesis se centra en los procesos de transporte y trata sobre el problema de enrutamiento de arcos capacitados (CARP) y su variante dinámica (DCARP). Introduje el primer operador de movimiento de tamaño de paso medio para el CARP, el operador de plan de subrutas, diseñado para optimizar los planes de rutas de manera más efectiva. Las pruebas demostraron que, en promedio, supera a los operadores existentes en la búsqueda de mejores planes de servicio. Otra contribución significativa de mi trabajo es el desarrollo del primer algoritmo de colonia de abejas artificiales (ABC) para resolver el CARP. Propuse dos versiones: una versión enfocada en la exploración para el CARP y una versión enfocada en el descubrimiento para el DCARP. Para evaluar el rendimiento de los solucionadores de DCARP en entornos simulados realistas, desarrollé un marco de trabajo basado en datos para DCARP capaz de manejar todos los posibles eventos dinámicos. Este marco actualiza la instancia del problema y el plan de servicio procesando eventos del módulo de generación de eventos o de un software de simulación de tráfico, e invoca su módulo de reprogramación cuando es necesario. Como parte del marco, propuse un algoritmo de reruteo mínimo (RR1), que garantiza la viabilidad con modificaciones mínimas en el plan de servicio. Los experimentos realizados sobre instancias de CARP basadas en problemas reales mostraron que el algoritmo CARP-ABC enfocada en la exploración puede proporcionar planes de servicio casi óptimos en un corto período de tiempo. Los experimentos realizados sobre instancias de DCARP utilizando el marco de trabajo basado en datos mostraron que la combinación del algoritmo RR1 y el algoritmo CARP-ABC enfocada en el descubrimiento proporciona los mejores planes de servicio en la mayoría de los casos.

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Starkné dr. Werner Ágnesnek a folyamatos támogatásáért, türelméért, motivációjáért és szakmai segítségéért, amelyek a szakdolgozatom készítésének kezdetétől a disszertációm elkészítésének befejezéséig végigkísértek.

Köszönettel tartozom kollégáimnak, hallgatótársaimnak, a Villamosmérnöki és Információs Rendszerek Tanszéknek, valamint a Műszaki Informatikai Karnak a rengeteg támogatásért és segítségért, valamint a barátságos és kellemes környezet biztosításáért. Köszönöm a doktori iskola vezetőinek, dr. Hartung Ferenc professzor úrnak és dr. Hangos Katalin professzor asszonynak, valamint a doktori iskola titkárainak, dr. Dulai Tibornak és dr. Görbe Péternek a doktori tanulmányaim zökkenőmentes előrehaladásához nyújtott segítséget.

Külön köszönettel tartozom a doktori disszertációm bírálóinak, akik részletes visszajelzéseikkel és konstruktív javaslataikkal nagyban hozzájárultak a dolgozatom szakmai színvonalának emeléséhez.

Végezetül, de nem utolsósorban szeretnék köszönetet mondani a családomnak, különösen a szüleimnek, akik mindvégig kitartóan támogattak, bátorítottak és hittek bennem.

Tartalomjegyzék

Ábrák jegyzéke

Táblázatok jegyzéke

Algoritmusok jegyzéke

Jelölések

Általános

$\mathbb{P}(X)$	az X halmaz hatványhalmaza, $\mathbb{P}(X) = \{x \mid x \subseteq X\}$
$\mathbb{B}(X)$	az X halmaz feletti összes multihalmaz halmaza
$\varnothing$	üres érték, ha valaminek az értéke nincs megadva, akkor ezt az értéket veszi fel; nem összetévesztendő az üres halmazzal, amit a $\emptyset$ jelöl

A MOCC módszer

Esemény és eseményfolyam

$\mathcal{C}$	az esetazonosítók univerzuma
$\mathcal{A}$	a megfigyelhető tevékenységek univerzuma
$\mathcal{U}$	az értékek univerzuma
$\mathcal{U}^n$	n darab esemény attribútum értékeinek univerzuma, $\mathcal{U}^n = \prod_{i=1}^{n} \mathcal{U}_i$
$\mathcal{U}_i$	az i-edik esemény attribútum értékeinek univerzuma $\mathcal{U}^n$-en belül, $\mathcal{U}_i \in \mathbb{P}(\mathcal{U})$
$\mathcal{E}$	a lehetséges események univerzuma, $\mathcal{E} = \mathcal{C} \times \mathcal{A} \times \mathcal{U}^n$
E	eseményfolyam, $E \in \mathcal{E}^\star$
c	esetazonosító, $c \in \mathcal{C}$
a	megfigyelhető tevékenység, $a \in \mathcal{A}$
u^n	n darab esemény attribútum értéknek a rendezett n-ese, $u^n = (u_1, u_2, \ldots, u_n) \in \mathcal{U}^n$
u_i	az i-edik esemény attribútum értéke u^n-en belül
e	esemény, $e = (c, a, u^n) \in \mathcal{E}$
S_l	a lehetséges eseménynapló lépések halmaza, egy adott $c \in \mathcal{C}$ esethez tartozó megfigyelhető események alapján, $S_l \subseteq \mathcal{A} \times \mathcal{U}^n$
s_l	eseménynapló lépés, egy adott $c \in \mathcal{C}$ esethez tartozó megfigyelt esemény alapján, $s_l = (a, u^n) \in S_l$
σ	nyomvonal, megfigyelt eseménynapló lépések véges sorozata, egy adott $c \in \mathcal{C}$ esethez tartozó megfigyelt események alapján, $\sigma \in S_l^\star$
π_i	projekciós függvény, egy rendezett n-es (pl. u^n vagy e) vagy sorozat (pl. σ) i-edik elemének a kinyeréséhez

Adat Petri-háló

P	a helyek véges halmaza
T	az átmenetek véges halmaza
F	irányított élek véges halmaza, ami a helyek és az átmenetek közötti kapcsolatot írja le, $F \subseteq (P \times T) \cup (T \times P)$
M	állapot jelölés, $M \in \mathbb{B}(P)$
M_i	kezdeti állapot jelölés, $M_i \in \mathbb{B}(P)$
M_f	végső állapot jelölés, $M_f \in \mathbb{B}(P)$
V	a folyamatváltozók (elnevezéseinek) véges halmaza
V'	a prímváltozók véges halmaza, $V' = \{v' \mid v \in V\}$
val	egy $v \in V$ folyamatváltozó (potenciálisan) végtelen elemszámú értékhalmazát visszaadó függvény, $val : V \to \mathbb{P}(\mathcal{U})$
$EXPR(V)$	az összes lehetséges $v \in V$ folyamatváltozóval definiált logikai kifejezések univerzuma
$expr$	(lineáris) őrkifejezés, $v \in V$ folyamatváltozókkal és $v' \in V'$ prímváltozókkal definiált logikai kifejezés, $expr \in EXPR(V \cup V')$
gd	adatfüggő őr, egy $t \in T$ átmenethez tartozó $expr \in EXPR(V \cup V')$ őrkifejezést visszaadó függvény, $gd : T \to EXPR(V \cup V')$
$eval_{V,val}$	őrkifejezés kiértékelő függvény, egy $expr \in EXPR(V \cup V')$ őrkifejezés igazságértékét értékeli ki, $eval_{V,val} : (EXPR(V \cup V') \times W) \to \{\texttt{true}, \texttt{false}\}$
in	egy $v \in V$ folyamatváltozó kezdeti értékét visszaadó függvény, $in : V \to \mathcal{U}$
wr	egy $t \in T$ átmenethez előírt változóírási műveletek által érintett folyamatváltozók halmazát visszaadó függvény, $wr : T \to \mathbb{P}(V)$
N	adat Petri-háló, ami egy (P, T, F) Petri-háló további perspektívákat leíró (V, val, in, wr, gd) komponensekkel, $N = (P, T, F, V, val, in, wr, gd)$
(M, α)	egy adat Petri-háló állapota, ahol M a (P, T, F) Petri-háló jelölése és α a jelenlegi folyamatváltozó hozzárendelés, $M \in \mathbb{B}(P)$, $\alpha \in \mathcal{U}^{\lvert V \rvert}$
τ	láthatatlan átmenet címkéje, olyan $t \in T$ átmenet címkéje, amihez nem tartozik tevékenységcímke
λ_a	tevékenységcímke függvény, ami minden $t \in T$ átmenethez visszaadja a megfelelő tevékenységet, ha a tevékenység megfigyelhető (azaz $a \in \mathcal{A}$), különben τ-t ad vissza, $\lambda_a : T \to \mathcal{A} \cup \{\tau\}$
λ_v	változócímke függvény, ami minden $v \in V$ folyamatváltozóhoz visszaadja a megfelelő esemény attribútum $0 < i \leq n$ index értéket, ami az u_i értéket azonosítja u^n-en belül, $\lambda_v : V \to \mathbb{Z}^+_{\leq n}$
N_λ	címkézett adat Petri-háló, egy N adat Petri-háló egyedi $p_i \in P$ forrás és $p_f \in P$ nyelő hellyel, valamint egy λ_a tevékenység és egy λ_v változócímke függvénnyel kiegészítve, $N_\lambda = (P, T, F, V, val, in, wr, gd, [p_i], [p_f], \lambda_a, \lambda_v)$

W	a lehetséges érvényes változó hozzárendelések halmaza, $W = \{w \in V \nrightarrow \mathcal{U} \mid \forall_{v \in dom(w)}(w(v) \in val(v))\}$
w	egy $t \in T$ átmenet tüzelése utáni új változó hozzárendelések, amiben csak az átmenet által írható változók szerepelnek, $w \in W$ és $dom(w) = wr(t)$
S_m	az érvényes modell lépések halmaza, $S_m \subset T \times W$
s_m	(érvényes) modell lépés, ami a tüzelt $t \in T$ átmenet és a $w \in W$ új változó hozzárendelések rendezett kettese, $s_m = (t, w) \in S_m$
ρ	tüzelési szekvencia, az N (vagy N_λ címkézett) adat Petri-hálón visszajátszható modell lépések véges sorozata, $\rho \in S_m^\star$

Igazítások

Γ_c	az összes megengedett vezérlésfolyam igazítási mozgás halmaza, $\Gamma_c = (\mathcal{A} \cup \{\gg\}) \times (T \cup \{\gg\}) \setminus \{(\gg, \gg)\}$
$(a, \gg)$	eseménynapló mozgás, olyan vezérlésfolyam igazítási mozgás, ami csak egy $a \in \mathcal{A}$ tevékenységből áll
$(\gg, t)$	modell mozgás, olyan vezérlésfolyam igazítási mozgás, ami csak egy $t \in T$ átmenetből áll
(a, t)	szinkron mozgás, olyan vezérlésfolyam igazítási mozgás, ami egy $a \in \mathcal{A}$ tevékenységből és egy $t \in T$ átmenetből áll, ahol $a = \lambda_a(t)$
γ_c	vezérlésfolyam igazítás, vezérlésfolyam igazítási mozgások véges sorozata, $\gamma_c \in \Gamma_c^\star$
Γ	az összes megengedett több perspektívás igazítási mozgás halmaza, $\Gamma = (S_l \cup \{\gg\}) \times (S_m \cup \{\gg\}) \setminus \{(\gg, \gg)\}$
$(s_l, \gg)$	eseménynapló mozgás, olyan több perspektívás igazítási mozgás, ami csak egy $s_l \in S_l$ eseménynapló lépésből áll
$(\gg, s_m)$	modell mozgás, olyan több perspektívás igazítási mozgás, ami csak egy $s_m \in S_m$ modell lépésből áll
(s_l, s_m)	szinkron mozgás, olyan több perspektívás igazítási mozgás, ami egy $s_l = (a, u^n) \in S_l$ eseménynapló lépésből és egy $s_m = (t, w) \in S_m$ modell lépésből áll, ahol $a = \lambda_a(t)$; ha minden $v \in wr(t)$ esetén $\pi_{\lambda_v(v)}(u^n) = w(v)$, akkor helyes szinkron mozgás, különben helytelen szinkron mozgás
γ	több perspektívás igazítás, több perspektívás igazítási mozgások véges sorozata, $\gamma \in \Gamma^\star$
f_c	egy több perspektívás igazítási mozgást vezérlésfolyam igazítási mozgássá alakító függvény, $f_c : \Gamma \to \Gamma_c$
$f_c^\star$	egy több perspektívás igazítást vezérlésfolyam igazítássá alakító függvény, $f_c^\star : \Gamma^\star \to \Gamma_c^\star$
κ_a	tevékenységköltség-függvény, ami minden egyes megengedett vezérlésfolyam igazítási mozgáshoz egy nem negatív költséget rendel, $\kappa_a : \Gamma_c \to \mathbb{R}_{\geq 0}$

κ_v	változóköltség-függvény, ami minden egyes folyamatváltozó elnevezés, eseménynapló szerinti érték és modell szerinti érték rendezett hármashoz egy nem negatív költséget rendel, $\kappa_v : V \times \mathcal{U} \times \mathcal{U} \to \mathbb{R}_{\geq 0}$
K_c	a κ_a tevékenységköltség-függvény alapján egy γ_c vezérlésfolyam igazítás költségét kiszámító függvény, $K_c : \Gamma_c^\star \to \mathbb{R}_{\geq 0}$
K	a κ_a tevékenységköltség-függvény és a κ_v változóköltség-függvény alapján egy γ több perspektívás igazítás költségét kiszámító függvény, $K : \Gamma^\star \to \mathbb{R}_{\geq 0}$

OVAP és annak gyorsítótárazása

VAR	MILP változók halmaza
MAP	a $v \in V$ folyamatváltozóhoz vagy a $v' \in V'$ prímváltozóhoz $v_i \in VAR$ MILP változót rendelő függvény, $MAP : V \cup V' \nrightarrow VAR$
v_i	MILP változó a $v \in V$ folyamatváltozó i-edik modell lépéshez tartozó írási műveletéhez, $v_i \in VAR$
$\hat{v}_i$	Boolean MILP változó a $v \in V$ folyamatváltozó i-edik modell lépéshez tartozó írási műveletéhez, ami a megfigyelt attribútumérték megvalósíthatósági mutatója, $\hat{v}_i \in VAR$
$CSTR$	MILP korlátozások halmaza
OBJ	MILP objektív függvény
Ω	OVAP gyorsítótár, $\Omega : (T \times W)^\star \to W^\star \times \mathbb{R}_{\geq 0} \times W$
$\overline{\rho}$	változó írási szekvencia, érvényes változó hozzárendeléseket tartalmazó modell lépések véges sorozata, ami egy OVAP azonosítására szolgál, $\overline{\rho} \in (T \times W)^\star$
ω	egy OVAP megoldása, érvényes változó hozzárendelések véges sorozata, $\omega \in W^\star$

Inkrementális több perspektívás előtag-igazítás számítás szinkron szorzat hálóval

N^S	szinkron szorzat háló (SPN), $N^S = (P^S, T^S, F^S, M_i^S, M_f^S, \lambda_a^S)$
t^σ	átmenet az SPN eseménynapló részében
t	átmenet az SPN modell részében
$\mathcal{R}(N^S, M_i^S)$	az N^S SPN összes elérhető jelölése (más néven állapottere) egy M_i kezdeti állapot jelölés esetén
m	az N^S SPN egy lehetséges jelölése (állapota), $m \in \mathcal{R}(N^S, M_i^S)$
$\mathcal{D}_x$	egy adott $c \in \mathcal{C}$ esetazonosítóhoz tartozó gyorsítótárazott objektumok, ahol x egy adott objektumot jelöl, $x \in \{\sigma, O, C, g, \mu, \alpha, \overline{\gamma}, \gamma\}$
O	nyitott halmaz, a még megvizsgálandó állapotok halmaza, $O \subseteq \mathcal{R}(N^S, M_i^S)$
C	zárt halmaz, a már megvizsgált állapotok halmaza, $C \subseteq \mathcal{R}(N^S, M_i^S)$

f	egy $m \in \mathcal{R}(N^S, M_i^S)$ állapothoz a teljes becsült költséget (a g addig ismert költség és a h becsült fennmaradó költség összegét) visszaadó függvény, $f : \mathcal{R}(N^S, M_i^S) \to \mathbb{R}_{\geq 0}$
h	heurisztikus függvény, ami minden $m \in \mathcal{R}(N^S, M_i^S)$ állapothoz visszaadja a becsült fennmaradó költséget, $h : \mathcal{R}(N^S, M_i^S) \to \mathbb{R}_{\geq 0}$
g	eddigi költség függvény, egy $m \in \mathcal{R}(N^S, M_i^S)$ SPN állapothoz az addig ismert teljes költséget visszaadó függvény, $g : \mathcal{R}(N^S, M_i^S) \to \mathbb{R}_{\geq 0}$
μ	előzmény függvény, ami egy $m' \in \mathcal{R}(N^S, M_i^S)$ állapotba vezető $(t^\sigma, t) \in T^S$ átmenet és $m \in \mathcal{R}(N^S, M_i^S)$ állapot rendezett párját visszaadó függvény, $\mu : \mathcal{R}(N^S, M_i^S) \to T^S \times \mathcal{R}(N^S, M_i^S)$
α	változó értékek függvény, egy $m \in \mathcal{R}(N^S, M_i^S)$ állapothoz a folyamatváltozók aktuális értékét visszaadó függvény, $\alpha : \mathcal{R}(N^S, M_i^S) \to W$
$\overline{\gamma}$	igazítás függvény, egy $m \in \mathcal{R}(N^S, M_i^S)$ állapothoz az aktuális $\gamma \in \Gamma^\star$ több perspektívás előtag-igazítást visszaadó függvény, $\overline{\gamma} : \mathcal{R}(N^S, M_i^S) \to \Gamma^\star$
g_a	eddigi tevékenységköltség, az éppen vizsgált $m \in \mathcal{R}(N^S, M_i^S)$ állapothoz az addig ismert tevékenységköltséget tároló változó, $g_a \in \mathbb{R}_{\geq 0}$
g_v	eddigi változóköltség, az éppen vizsgált $m \in \mathcal{R}(N^S, M_i^S)$ állapothoz az addig ismert változóköltséget tároló változó, $g_v \in \mathbb{R}_{\geq 0}$

Eset gyorsítótárazó módszer

χ	frissítési ráta (másodpercben megadva), $\chi \in \mathbb{Z}_{>0}$
t_{most}	az aktuális időpont
$t_{\max}$	esetfrissítési határidő (másodpercben megadva), $t_{\max} \in \mathbb{Z}_{>0}$
$l_{\max}^{cc}$	a befejezett esetek gyorsítótárának maximális mérete, $l_{\max}^{cc} \in \mathbb{Z}_{>0}$
n	a folyamatban lévő esetek gyorsítótárának az aktuális mérete, $n \in \mathbb{Z}_{\geq 0}$
k	a befejezett esetek gyorsítótárának az aktuális mérete, $k \in \mathbb{Z}_{\geq 0}$

Statikus és dinamikus CARP probléma

(D)CARP

V	csúcsok véges halmaza
v_0	telephely(csúcs), $v_0 \in V$
A	ívek (irányított élek) véges halmaza, $A \subseteq V \times V$
G	irányított gráf, $G = (V, A)$
T	feladathalmaz, ívek részhalmaza, $T \subseteq A$

t_0	álfeladat, ami egy hurokél, $t_0 = (v_0, v_0)$
$head$	egy $a \in A$ ív fejcsúcsát visszaadó függvény, $head : A \to V$
$tail$	egy $a \in A$ ív végcsúcsát visszaadó függvény, $tail : A \to V$
inv	egy $a \in A$ ív inverzét visszaadó függvény, $inv : A \to A$
dc	egy $a \in A$ ív áthaladási költségét visszaadó függvény, $dc : A \to \mathbb{R}_{\geq 0}$
id	egy $t \in T$ feladat egyedi azonosítóját visszaadó függvény, $id : T \to \mathbb{Z}_{\geq 0}$
dem	egy $t \in T$ feladat igényét visszaadó függvény, $dem : T \to \mathbb{R}_{>0}$
sc	egy $t \in T$ feladat szolgáltatási költségét visszaadó függvény, $sc : T \to \mathbb{R}_{\geq 0}$
mdc	két $v_i, v_j \in V$ csúcs közötti minimális teljes áthaladási költséget visszaadó függvény, $mdc : V \times V \to \mathbb{R}_{\geq 0}$
n	a végrehajtandó feladatok száma, $n \in \mathbb{Z}_{>0}$
w	a járművek száma, $w \in \mathbb{Z}_{>0}$
q	egy jármű maximális kapacitása, $w \in \mathbb{R}_{>0}$
$\mathfrak{I}$	a lehetséges CARP példák halmaza
I	egy CARP példa, $I = (V, v_0, A, T, n, w, q, head, tail, dc, id, dem, sc, inv, mdc) \in \mathfrak{I}$
$\mathcal{R}$	a lehetséges útvonaltervek univerzuma, $\mathcal{R} \subset \mathbb{P}((T \cup \{t_0\})^\star)$
S	szolgáltatási terv, az I CARP példa egy lehetséges megoldása, $S = \{r_1, r_2, \ldots, r_{\lvert S \rvert}\} \subset \mathcal{R}$
r_k	az S szolgáltatási terv k-adik útvonalterve, $r_k = \langle t_0, t_{k,1}, t_{k,2}, \ldots, t_{k,l_k}, t_0 \rangle \in (T \cup \{t_0\})^\star$
l_k	az S szolgáltatási terv k-adik útvonaltervében (azaz az r_k útvonaltervben) szereplő feladatok száma, $l_k = \lvert r_k \rvert - 2$
$t_{k,i}$	az S szolgáltatási terv k-adik útvonaltervének az i-edik feladata, $1 \leq i \leq l_k$ és $t_{k,i} \in T$
DC	egy I (D)CARP példához kapcsolódó r_k útvonalterv teljes áthaladási költségét visszaadó függvény, $DC : \mathfrak{I} \times (T \cup \{t_0\})^\star \to \mathbb{R}_{>0}$
SC	egy I (D)CARP példához kapcsolódó r_k útvonalterv teljes szolgáltatási költségét visszaadó függvény, $SC : \mathfrak{I} \times (T \cup \{t_0\})^\star \to \mathbb{R}_{>0}$
TC	egy I (D)CARP példához kapcsolódó S szolgáltatási terv teljes költségét visszaadó függvény, $TC : \mathfrak{I} \times \mathcal{R} \to \mathbb{R}_{>0}$
LB	egy I CARP példa megoldásának teljes költségének alsó korlátát visszaadó függvény, ami az összes feladat szolgáltatási költségének az összege, $LB : \mathfrak{I} \to \mathbb{R}_{\geq 0}$
T_v	virtuális feladatok halmaza, $T_v \subseteq T$
t_{v_k}	a $k \in R$ azonosítójú jármű virtuális feladata, $t_{v_k} \in T_v$
H	az összes jármű azonosítójának halmaza, $\lvert H \rvert = w$
H_e	a (jelenleg) hibás járművek azonosítóinak halmaza, $H_e \subseteq H$
H_f	a (jelenleg) szabad járművek azonosítóinak halmaza, $H_f \subseteq H$
R	az összes útvonalterv azonosítójának halmaza

R_e azon útvonaltervek azonosítóinak halmaza, amelyek végrehajtása leállt, $R_e \subseteq R$

rt a $k \in R$ azonosítójú útvonalterv virtuális feladatát visszaadó függvény, $rt : R \to T_v$

rv a $k \in R$ azonosítójú útvonaltervet végrehajtó jármű azonosítóját visszaadó függvény, $rv : R \to H$

m a DCARP példák száma egy DCARP forgatókönyvben, $m \in \mathbb{Z}_{\geq 0}$

$\mathcal{I}$ egy DCARP forgatókönyv, DCARP példák véges halmaza, $\mathcal{I} = \langle I_0, I_1, \ldots, I_{m-1} \rangle$

I_i az i-edik DCARP példa egy DCARP forgatókönyvben, $0 \leq i < m$ és $I_i = (V, v_0, A, T, T_v, n, q, H, H_e, H_f, R, R_e, rt, rv, head, tail, dc, inv, dem, sc, mdc)$

S_i az i-edik DCARP példa elfogadott szolgáltatási terve, hasonló S-hez

Részútvonalterv művelet

p_{rc} az újrakapcsolódás valószínűsége, $p_{rc} \in [0, 1]$

p_{cp} a korrekció és a perturbáció valószínűsége, $p_{cp} \in [0, 1]$

p_l a linearitás valószínűsége, $p_l \in [0, 1]$

p_{rnd} egy véletlen valószínűségi érték, $p_{rnd} \in [0, 1]$

$l_{\min}$ a részútvonalterv minimális hossza, $l_{\min} \in \mathbb{Z}_{>0}$ és $l_{\min} \leq l_{\max}$

$l_{\max}$ a részútvonalterv maximális hossza, $l_{\max} \in \mathbb{Z}_{>0}$ és $l_{\min} \leq l_{\max}$

l a részútvonalterv kiválasztott hossza, $l_{\min} \leq l \leq l_{\max}$

$t_{k,s}$ a részútvonalterv kiválasztott kezdő feladata, $t_{k,s} \in T$

$t_{k,e}$ a részútvonalterv kiválasztott befejező feladata, $t_{k,e} \in T$

$n_{\max}$ egy feladat szomszédságának (maximális) mérete, $n_{\max} \in \mathbb{Z}_{>0}$

$t_{k,s^\star}$ a $t_{k,s}$ feladat környezetéből kiválasztott feladat, $t_{k,s^\star} \in T$

$t_{k,e^\star}$ a $t_{k,e}$ feladat környezetéből kiválasztott feladat, $t_{k,e^\star} \in T$

$r_k^\star$ a kiválasztott részútvonalterv az $r_k \in S$ útvonalterven belül

$r_k^\#$ a csonkított útvonalterv, az $r_k \in S$ útvonalterv az $r_k^\star$ részútvonalterv nélkül

r'_k a kapott útvonalterv

CARP-ABC algoritmusok

n_{cs} a kolónia mérete, $n_{cs} \in \mathbb{Z}_{>0}$

n_{mi} az iterációk maximális száma, $n_{mi} \in \mathbb{Z}_{>0}$

n_{gsl} a globális keresési limit, $n_{gsl} \in \mathbb{Z}_{>0}$

n_{lsl} a lokális keresési limit, $n_{lsl} \in \mathbb{Z}_{>0}$

n_{sal} a megoldás korhatára (a populáción belül), $n_{sal} \in \mathbb{Z}_{>0}$

C az aktuális kolónia, a megoldások (szolgáltatási tervek) egy halmaza, $C = \{S_1, S_2, \ldots, S_{n_{cs}}\}$

$\overline{C}$ az előző kolónia

α_i	egy S_i megoldás kora, $\alpha_i \in \mathbb{Z}_{>0}$
p_i	egy S_i megoldás valószínűségi értéke, $p_i \in [0,1]$
$\mathcal{A}$	a C kolónián belüli megoldások korának halmaza, $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_{n_{cs}}\}$
P	a C kolónián belüli megoldások valószínűségi értékeinek halmaza, $P = \{p_1, p_2, \ldots, p_{n_{cs}}\}$, $\sum_{i=1}^{n_{cs}} p_i = 1$
$S^\star$	a jelenleg ismert globálisan legjobb megoldás
S'	egy iteráción belül talált legjobb megoldás
$S_i^\star$	az $S_i \in C$ megoldás szomszédságában talált legjobb megoldás
S_i'	az $S_i \in C$ megoldás szomszédságában, egy iteráción belül talált legjobb megoldás
$S_{i,j}^\star$	az $S_{i,j}$ megoldás ($S_i \in C$ megoldás szomszédja) szomszédságában talált legjobb megoldás
$S_{i,j}'$	az $S_{i,j}$ megoldás ($S_i \in C$ megoldás szomszédja) szomszédságában, egy iteráción belül talált legjobb megoldás
$\alpha^\star$	az $S^\star$ megoldás aktuális kora
α'	az S megoldás aktuális kora
$\alpha_i^\star$	az $S_i^\star$ megoldás aktuális kora
α_i'	az S_i' megoldás aktuális kora
n_{nil}	nincs javulási korlát a megfigyelő méh fázisban, $n_{nil} \in \mathbb{Z}_{>0}$
O	mozgási műveletek függvényét tartalmazó sorozat
o	egy mozgási művelet függvénye, $o \in O$
T_i	az $S_i^\star$ megoldásból kinyert (nem ál)feladatok sorozata, $T_i \subseteq T$
L_{t_1}	a $t_1 \in T_i$ megoldáshoz létrehozott jelölt lista, $L_{t_1} \subset T_i$

DCARP keretrendszer és RR1 algoritmus

e	esemény, $e \in \mathbb{Z}_{>0} \times A \cup \{-1\} \times H \cup \{-1\} \times \{-1,0,1\} \times \mathbb{R}_{\geq 0} \cup \{-1\} \times \mathbb{R}_{\geq 0} \cup \{-1\} \times \mathbb{R}_{\geq 0} \cup \{-1\} \times \{-1,0,1\}$
ts	az esemény időbélyege, $ts \in \mathbb{Z}_{>0}$
a	az eseményhez kapcsolódó ív (útszakasz), $a \in A \cup \{-1\}$
h	az eseményhez kapcsolódó jármű azonosítója, $h \in H \cup \{-1\}$
α	a tevékenység típusát jelölő érték utazási vagy szolgáltatási esemény esetén, $\alpha \in \{-1,0,1\}$, ahol a -1 üres értéket, a 0 utazási tevékenységet, az 1 pedig szolgáltatási tevékenységet jelöl
dc_a	az esemény által érintett $a \in A$ ív új áthaladási költsége, $dc_a \in \mathbb{R}_{\geq 0} \cup \{-1\}$
sc_a	az esemény által érintett $a \in A$ ív(feladat) (új) szolgáltatási költsége, $sc_a \in \mathbb{R}_{\geq 0} \cup \{-1\}$
dem_a	az esemény által érintett $a \in A$ ív(feladat) (új) igénye, $dem_a \in \mathbb{R}_{\geq 0} \cup \{-1\}$

h_e az esemény által érintett $h \in H$ jármű állapotváltozását jelölő érték, $h_e \in \{-1, 0, 1\}$, ahol a -1 üres értéket, a 0 beüzemelést, az 1 pedig meghibásodást jelöl

T_k az aktuális esemény által érintett feladatok sorozata, $t \in T \forall t \in T_k$

dem_{T_k} a $t \in T_k$ feladatok teljes igénye, $dem_{T_k} \in \mathbb{R}_{>0}$

$\overline{S}$ szolgáltatási terv, ami nem tartalmazza az aktuális esemény által érintett eseményeket (azaz T_k-t), $\overline{S} \in \mathcal{R}$

S' az új szolgáltatási terv, $S' \in \mathcal{R}$

tc' az új szolgáltatási terv teljes költsége, $tc' \in \mathbb{R}_{>0}$

$load$ egy I (D)CARP példa i-edik útvonaltervére (r_i) vonatkozó terhelést (az-az az abban szereplő feladatok igényeinek összegét) visszaadó függvény, $load : \mathfrak{I} \times (T \cup \{t_0\})^\star \to \mathbb{R}_{>0}$

ids a T_k feladatokat befogadni képes $i \in R$ útvonalterv azonosítók halmaza, $ids \subseteq R$

S_i potenciális új szolgáltatási terv, ahol a T_k feladat(ok) az $\overline{S}$ szolgáltatási tervben az $i \in ids$ azonosítójú útvonaltervbe van(nak) beszúrva, $S_i \in \mathcal{R}$

j_a az $a \in T$ feladat legjobb beillesztési pozíciója az $r_i \in S_i$ útvonalterven belül, $j_a \in \{1, 2, \ldots, |r_i| - 1\}$

inv_a az $a \in T$ feladat legjobb kiszolgálási irányát jelölő érték, $inv_a \in \{true, false\}$, ahol a $true$ érték azt jelenti, hogy a helyett $inv(a)$-t érdemes beszúrni a szolgáltatási tervbe (feltéve, hogy $a \in dom(inv)$)

tc_{ix} az S_i szolgáltatási terv teljes költsége, ha az a feladat ($x = 1$ esetén) vagy az $inv(a)$ feladat ($x = 2$ esetén, ha $a \in dom(inv)$) az éppen vizsgált j pozícióba van beszúrva az $r_i \in S_i$ útvonaltervben, $tc_{ix} \in \mathbb{R}_{>0}$, ahol $x \in 1, 2$

Δtc_{ix} az S_i szolgáltatási terv teljes költségének a változása, ha az a feladat ($x = 1$ esetén) vagy az $inv(a)$ feladat ($x = 2$ esetén, ha $a \in dom(inv)$) az éppen vizsgált j pozícióba van beszúrva az $r_i \in S_i$ útvonaltervben, $tc_{ix} \in \mathbb{R}$, ahol $x \in 1, 2$

tc_i az S_i szolgáltatási terv teljes költsége, $tc_i \in \mathbb{R}_{>0}$

Δtc_i az S_i szolgáltatási terv teljes költségének a változása, ha az a vagy az $inv(a)$ feladat (inv_a értékétől függően) a j_a pozícióba van beszúrva az $r_i \in S_i$ útvonaltervben, $tc_i \in \mathbb{R}$

Rövidítések

Több perspektívás online megfelelőség-ellenőrzés

BMCC	Balanced Multi-perspective Conformance Checking
CCaaS	Online Conformance Checking as a Service
CoCoMoT	Computing Conformance Modulo Theories
CRUD	Create Read Update Delete
DPN	Data Petri Net
HMM	Hidden Markov Model
ILP	Integer Linear Programming
MILP	Mixed-Integer Linear Programming
MOCC	Multi-perspective Online Confromance Checking
OCC	Online Confromance Checking
OVA	Optimal Variable Assignment
OVAP	Optimal Variable Assignment Problem
SMT	Satisfiability Modulo Theories
SPN	Synchronous Product Net

Statikus és dinamikus kapacitáskorlátos ív útvonaltervezési probléma

ABC	Artificial Bee Colony
ACOPR	Ant Colony Optimization algorithm with Path Relinking
ARP	Arc Routing Problem
CABC	Combinatorial Artificial Bee Colony Algorithm
CARP	Capacitated Arc Routing Problem
DAG	Directed Acyclic Graph
DARP	Dynamic Arc Routing Problem
DCARP	Dynamic Capacitated Arc Routing Problem
GSTM	Greedy Sub Tour Mutation
HMA	Hybrid Metaheuristic Approach
HyLS	Hybrid Local Search

MAENS	Memetic Algorithm with Extended Neighborhood Search
qABC	quick Artificial Bee Colony
qCABC	quick Combinatorial Artificial Bee Colony Algorithm
RPSH	Randomized Path-Scanning Heuristic
RSG	Random Solution Generation
RTS	Repair-based Tabu Search
SUMO	Simulation of Urban Mobility
TSA	Tabu Search Algorithm
TSP	Traveling Salesman Problem
VRP	Vehicle Routing Problem

Megjegyzés: Ha egy rövidítésben zárójeles rész szerepel, az azt jelenti, hogy az adott állítás a zárójeles szövegrésszel rendelkező és az anélküli kifejezésre is vonatkozik. Például a „(D)CARP" esetében az állítás mind a DCARP, mind a CARP rövidítésű kifejezésre érvényes.

1. fejezet

Bevezetés

A doktori kutatási munkám során üzleti folyamatokhoz (pontosabban gyártási és szállítási folyamatokhoz) kapcsolódó problémákkal foglalkoztam. A gyártási folyamatok esetében olyan fix folyamatokat feltételeztem, amik leginkább heterogén tevékenységekből állnak. Ezen problémákhoz a folyamatmodell és az eseményadatok adottak. A cél a folyamat online megfigyelése annak érdekében, hogy az elvárt működéstől való eltérések és azoknak okai minél hamarabb felderíthetők legyenek, ezáltal minimalizálva a hibás termékek számát és/vagy a leállások számát és hosszát. A szállítási folyamatok esetében rugalmas folyamatokat feltételeztem, amik leginkább homogén tevékenységekből állnak. Ezen problémákhoz a matematikai modell (úthálózat, feladatok halmaza stb.) és az eseményadatok adottak. A cél a folyamat online megfigyelése annak érdekében, hogy szükség esetén egy újragondolt, módosított szolgáltatási terv legyen létrehozva, a teljes szervizidő és/vagy a CO_2-kibocsátás minimalizálása érdekében. A dolgozatomban az „online" kifejezést olyan megoldásokra használom, amelyek folyamatos hálózati kapcsolat révén képesek adatcserére, valamint valós idejű feldolgozást biztosítanak.

A doktori tanulmányaim során végzett kutatási munkám két fő részre osztható:

1. **Több perspektívás online megfelelőség-ellenőrzés**: Ebben a részben a rövid átfutási idővel rendelkező folyamatok (pl. gyártási folyamatok) online megfigyelését és javítását támogató megoldások létrehozásával foglalkoztam. Ennek keretében kifejlesztettem egy több perspektívás online megfelelőség-ellenőrzési módszert, valamint két folyamatadat vizualizációs megoldást. A javasolt megoldások együttes használata lehetővé teszi a folyamatok hatékony online nyomonkövetését, és a folyamat hibás működése esetén az elvárt viselkedéstől való eltérések kiemelésével segítik annak javítását.

2. **Statikus és dinamikus kapacitáskorlátos ív útvonaltervezési probléma**: Ebben a részben a szállítási folyamatok online megfigyelését és javítását támogató megoldások létrehozásával foglalkoztam. Ennek keretében létrehoztam egy adat-vezérelt dinamikus kapacitáskorlátos ív útvonaltervezési probléma (DCARP) keretrendszert, ami megfelelő adatforrás esetén képes a szállítási folyamathoz felhasznált szolgáltatási tervet – a problémát módosító események bekövetkezését követően is – megvalósítható állapotban tartani. Ehhez szükség esetén módosítja a szolgáltatási tervet, amihez (D)CARP megoldó algoritmusokat használ. Újratervezés esetén az általam CARP-hoz kifejlesztett mesterséges méhcsalád (ABC) algoritmus és minimális újratervező (RR1) al-

goritmus együttes használata javasolt a minél jobb szolgáltatási terv mielőbbi megtalálása érdekében.

1.1. Több perspektívás online megfelelőség-ellenőrzés

A mai információs rendszerekben a végrehajtott üzleti folyamatokból nagy mennyiségű adat (úgynevezett eseményadat) keletkezik. A folyamatbányászat (*process mining*) az adattudomány és a folyamattudomány metszetének tekinthető. Célja a valós folyamatok javítása azáltal, hogy hasznosítható tudásanyagot nyer ki a folyamatok korábban rögzített lefutásaiból, a rendelkezésre álló eseményadatok felhasználásával. Kezdetben a folyamatbányászatnak három fő típusa volt: a folyamat felfedezés (*process discovery*), a megfelelőség-ellenőrzés (*conformance checking*) és a bővítés (*enhancement*) [1, 2]. Az elmúlt pár évben a bővítés helyett négy további típus jelent meg: a teljesítményelemzés (*performance analysis*), az összehasonlító folyamatbányászat (*comparative process mining*), a prediktív folyamatbányászat (*predictive process mining*), valamint a cselekvésorientált folyamatbányászat (*action-oriented process mining*) [3]. A dolgozatom ezek közül a megfelelőség-ellenőrzéssel és a prediktív folyamatbányászattal foglalkozik.

A megfelelőség-ellenőrzés [4] során egy folyamat eseményadatai kerülnek összehasonlításra ugyanannak a folyamatnak egy (kézzel készített vagy felfedezett) folyamatmodelljével, így a megfigyelt és a modellezett viselkedés közötti eltérések felderíthetők, lokalizálhatók és magyarázhatók. Az azonosított eltérések például egy automatizált gyártási folyamatban egy olyan gyártóeszközre mutathatnak, amely időnként meghibásodik (azaz az eszköz nem úgy működik, ahogyan az elvárható).

Napjainkban a gyakorlatban elfogadott alaptechnika a megfelelőség-ellenőrzési statisztikák kiszámítására az igazítások (*alignments*) kiszámítása [5], amely során minden egyes befejezett folyamatlefutás a hozzá leghasonlóbb folyamatmodellben lehetséges útvonalhoz kerül hozzáigazításra (feltételezve, hogy az került végrehajtásra vagy legalábbis azt próbálták végrehajtani) [4]. Az igazítások által pontosan megjelölt különbségeket a szakértők értelmezhetik, így következtetéseket tudnak levonni és intézkedéseket tudnak tenni a jövőbeli folyamat végrehajtások javítása érdekében.

Az elmúlt években több igazítás-alapú online megfelelőség-ellenőrzési megoldást is kifejlesztettek [6, 7], amelyek a még folyamatban lévő és emiatt hiányos folyamatlefutásokra is alkalmazhatók. Az online és offline környezetben végzett igazításszámítás közötti különbség az, hogy online környezetben a nyomvonal hiányossága nem számít hibás viselkedésnek, mivel a folyamatlefutás még folyamatban lehet. Emiatt online környezetben előtag-igazítások (*prefix-alignments*) kerülnek kiszámításra.

Problémafelvetés

Az elmúlt években számos különböző megfelelőség-ellenőrzési megoldást fejlesztettek ki, azonban még mindig **nincs olyan megoldás, amely lehetővé tenné a megfelelőség online elemzését több perspektíva figyelembevételével**. Az online megfelelőség-ellenőrzéssel kapcsolatos meglévő munkák (pl. [8, 9, 6, 7, 10]) csak a vezérlésfolyam (*control-flow*) perspektívára (azaz az események sorrendjére) koncentrálnak. A többperspektívás megfelelőség-ellenőrzéssel kapcsolatos munkák (pl. [11, 12, 13]) pedig csak utólagos elemzést tesznek lehetővé, így a nem megfelelő viselkedéseket csak a folyamat lefutás befejezése után fedezik fel. Ahogy az

a [4] publikációban is kiemelésre került, a létező megfelelőség-ellenőrzési megoldások túlnyomó többsége csak a vezérlésfolyam perspektívát veszi figyelembe, holott más perspektívák (pl. idő, erőforrás vagy adat) figyelembevétele is fontos lenne, hiszen az eltérések nem csak a tevékenységek sorrendjében lehetnek. Például, egy gyártási folyamat esetében nem elég tudni, hogy melyik műveletet hajtották végre, hogy megmondjuk, helyesen hajtották-e végre, a végrehajtás részleteinek (pl. felhasznált erőforrás) ismerete is szükséges. Emiatt az ilyen folyamatok többszempontú szemléletére van szükség. Továbbá, online környezetben a folyamatok végrehajtásában az eltérések (bármely perspektívában) szinte azonnal észlelhetők, így ellenintézkedések kezdeményezhetők az általuk okozott esetleges negatív hatások csökkentése érdekében.

A vizualizáció kulcsfontosságú a folyamatok nyomon követése szempontjából, mivel intuitív és átfogó ábrázolást biztosít az összetett folyamatadatokról, lehetővé téve az érdekelt felek számára az eltérések és azok kiváltó okainak gyors azonosítását. Az időbeli információk beépítése a vizualizációkba alapvető fontosságú, mivel az eseményeknek kontextust, sorrendet és időtartamot biztosít, elősegítve a dinamika, a függőségek és az időbeli előrehaladás mélyebb megértését. Számos folyamatbányászati szoftvereszköz áll rendelkezésre a folyamatadatok vizualizálásához, azonban azok **egyáltalán nem vagy nem megfelelően ábrázolják az időbeli információkat**. Az azonos időbélyeggel rendelkező események, az úgynevezett átfedő események különösen problémát jelentenek, és gyakran nem kezelik őket hatékonyan. Az általam kifejlesztett korábbi vizualizációs megoldások [14, 15, 16, 17] – más irodalomban felelhető vizualizációs megoldáshoz hasonlóan – képesek az eseményadatok online vizualizációjára az idő függvényében, de az átfedő eseményeket nem képesek kezelni. További problémát jelent, hogy **az eseményeket jellemzően vagy időpontokként, vagy időintervallumokként ábrázolják, és nincs olyan módszer, amely mindkettő ábrázolási módot használja**. A megfelelőség-ellenőrzés kimeneteire (pl. igazítási adatokra) bár létezik néhány vizualizációs megoldás, de egyik sem terjed ki az időbeli információkra. Ebből adódóan, **a megfelelőség-ellenőrző algoritmusok által kimutatott eltérések grafikus megjelenítése továbbra is nyitott kihívásnak számít** [18].

Célkitűzések

Az előző szakaszban kiemelt problémák megoldása érdekében a következő célokat tűztem ki:

- Egy olyan több perspektívás online megfelelőség-ellenőrzési **(MOCC) módszer** létrehozása, amely támogatja egy folyamat online nyomon követését különböző nézőpontokból.
- Olyan **folyamatadat vizualizációs módszerek** kidolgozása, amelyek
  - vegyes (fix és időtartamos) idővonal-alapú vizualizációt használnak;
  - megfelelően kezelik az átfedő objektumokat;
  - képesek az eredeti eseményadatok és az eseményadatokra kapott MOCC kimenetek (igazítási adatok) grafikus megjelenítésére.

A [19] publikáció szerint a folyamatelemzési vizualizációs módszerek a bemeneti adatok és a megjelenítendő információk alapján három fő kategóriába sorolhatók:

- folyamatmodell vizualizáció,
- eseményadat vizualizáció, és
- összekapcsolt vizualizáció.

Az eredeti eseményadatok vizualizációja az eseményadat-vizualizáció kategóriájába tartozik, míg az igazítási adatok vizualizációja az összekapcsolt vizualizáció kategóriájába sorolható.

Megoldások

Az első munkámban [20] egy viselkedési mintákon alapuló MOCC módszerrel próbálkoztam. A második munkámban [21] már egy **előtag-igazítás alapú MOCC módszer** kifejlesztését céloztam meg, aminek a megvalósítását a legutóbbi munkámban [22] tökéletesítettem. A létrehozott megoldás egy (adat Petri-háló formájában megadott) több perspektívás folyamatmodell és egy eseményfolyam megfigyelt eseményei közötti optimális több perspektívás előtag-igazításokat ad vissza. A javasolt megoldás használata rövid átfutási idővel és előírásos folyamatmodellel rendelkező folyamatok végrehajtásának nyomon követéséhez ajánlott. Ezzel megvalósítható egy alacsony költségű, ideiglenes felügyeleti rendszer az éretlen, még nem eléggé kiforrott folyamatok számára, amíg a folyamat el nem éri a kellően kiforrott állapotot. Emellett olyan régi folyamatokat is támogathat, ahol a felügyeleti rendszer nem képes minden rendellenességet felismerni, és a rendszert nem lehet módosítani. Ilyen esetekben az eltérések észlelésére kiegészítő megoldásként lehetne használni.

A probléma megoldásához **két folyamatadat vizualizációs módszer**t fejlesztettem ki: egyet az eredeti **eseményadatok vizualizálásához** és egyet **igazítási adatok vizualizálásához** [23]. Az igazítási adatokat az előtag-igazítás alapú MOCC módszer biztosítja az eseményadatok és egy adott (adat Petri-háló formájában megadott) folyamatmodell felhasználásával. Mindkettő módszert úgy terveztem meg, hogy magába foglalják a meglévő módszerek előnyös tulajdonságait, ugyanakkor kezeljék azok hiányosságait. A módszerek egy Gantt-diagram által inspirált vegyes (fix és idő-tartamos) idővonal-alapú vizualizációt használnak, amelyek az átfedő objektumokat is kezelik. Emellett lehetővé teszik a felhasználók számára, hogy kiválaszthassák, milyen perspektívából szeretnék vizsgálni az adatokat.

1.2. Statikus és dinamikus kapacitáskorlátos ív útvonaltervezési probléma

A kapacitáskorlátos ív útvonaltervezési probléma (*Capacitated Arc Routing Problem*, röviden *CARP*) egy kombinatorikus optimalizálási NP-nehéz probléma, amelyet először Golden és Wong mutatott be [24]. A CARP során meg kell határozni a járművek legkevesebb költséggel járó útvonalterveit (azaz egy szolgáltatási tervet) egy úthálózatot reprezentáló gráfon, bizonyos korlátozások mellett. A gráf csúcsok halmazát tartalmazza, amelyek élekkel és ívekkel (azaz irányított élekkel) vannak összekötve, amik útszakaszokat reprezentálnak. Minden útszakasznak van utazási költsége, és az útszakaszok egy adott részének járművek által kiszolgálandó igényei és szolgáltatási költségei is vannak. A szolgáltatási költség az adott útszakaszhoz tartozó igények kiszolgálásának a költsége, amely magában foglalja az utazási költséget is. Az igényeket tartalmazó éleket és íveket élfeladatoknak és ívfeladatoknak vagy egyszerűen

feladatoknak nevezzük. A CARP optimalizálásával a gráfhoz korlátozott kapacitással rendelkező járműveket rendelünk oly módon, hogy a lehető legalacsonyabb teljes költséggel szolgálják ki az összes feladatot. A problémának a valós világban számos alkalmazása van begyűjtési feladatoknál (pl. városi szilárd hulladékgyűjtés [25, 26]) és szétosztási feladatoknál (pl. téli utcák sózása [27, 28]) is.

A CARP az ív útvonaltervezési problémák (*Arc Routing Problem*, röviden *ARP*) családjába tartozik, amelynek számos változata van. Maga az ARP a jármű útvonaltervezési probléma (*Vehicle Routing Problem*, röviden *VRP*) egyik változata [29]. Az érdeklődőknek lehetőségük van az ARP-ok létező változatairól bővebben olvasni az annotált bibliográfiákban [30, 31], amelyek 2017-tel bezárólag összegyűjtötték és katalogizálták azokat a sajátosságaik szerint, illetve egy frissebb, 2021-ben megjelent irodalmi áttekintésben [32].

A normál CARP egy statikus problémát feltételez, amely távol áll a valóságtól. A szolgáltatási terv végrehajtása során olyan változások következhetnek be, amelyek módosíthatják a problémát, így hatással lehetnek az aktuális szolgáltatási terv megvalósíthatóságára és optimális mivoltára is [32]. Ha egy dinamikus esemény drasztikusan befolyásolja az aktuális szolgáltatási tervet (pl. új feladat jelenik meg, vagy egy jármű meghibásodik, mielőtt az útvonaltervében szereplő összes feladatot kiszolgálta volna), akkor a szolgáltatási terv újratervezésére van szükség. Emiatt a dinamikus kapacitáskorlátos ív útvonaltervezési probléma (*Dynamic Capacitated Arc Routing Problem*, röviden *DCARP*) – amely a CARP egy olyan változata, amely figyelembe veszi a szolgáltatási terv végrehajtása során bekövetkező dinamikus változásokat – jobb megközelítést kínál. Hasonlóképpen, az ARP dinamikus változatát dinamikus ív útvonaltervezési problémának (*Dynamic Arc Routing Problem*, röviden *DARP*) nevezzük.

Problémafelvetés

Mivel a CARP egy NP-nehéz probléma, a szakirodalomban az egzakt módszerek helyett elsősorban heurisztikákat és metaheurisztikákat (pl. [33, 34, 35, 36, 37]) használnak az optimális (vagy egy közel optimális) megoldás megtalálásához. **A meglévő CARP megoldók azonban vagy túl lassúak, vagy nem elég jó minőségű megoldásokat találnak.** A mesterséges méhcsalád (*Artificial Bee Colony*, röviden *ABC*) algoritmus [38] egy Karaboga által 2005-ben javasolt rajintelligencia alapú optimalizáló algoritmus, amit a mézelő méhraj intelligens táplálékkereső viselkedése inspirált. Ezt az algoritmust sikeresen alkalmazták több kombinatorikus optimalizálási problémára, amelyek hasonlóak a CARP-hoz [39, 40, 41, 42, 43, 44], és a [42] publikációban kimutatták, hogy **az ABC algoritmus jobb teljesítményt nyújt, mint a legtöbb evolúciós számításon alapuló optimalizálási algoritmus. Azonban, az ABC algoritmust korábban még soha nem alkalmazták (D)CARP-ra.**

A populáció alapú evolúciós algoritmusokban a populáció változatosságának növelése érdekében különböző lépésméretű mozgási műveleteket alkalmaznak az új, egymáshoz közeli megoldások létrehozásához. A CARP-hoz használt irodalomban fellelhető mozgási műveletek egyszerre csak nagyon kicsi változtatást (egy-kettő feladat pozíciója és/vagy kiszolgálási iránya változik) vagy nagyon nagy változtatást (egy vagy több útvonaltervben akár az összes feladat pozíciója és/vagy kiszolgálási iránya változik) végeznek (lásd 3.2.4. alszakasz), ami korlátozza a keresési tér felfedezését. Emiatt **szükség van közepes lépésméretű mozgási műveletekre**,

amelyek a kereső algoritmusokban egyensúlyt teremthetnének a helyi finomítás és a szélesebb körű felfedezés között, így potenciálisan jobb megoldásokat eredményezve.

Összegyűjtöttem és elemeztem a lehetséges eseményeket egy DCARP példára és annak megoldására (azaz szolgáltatási tervére) gyakorolt hatásuk alapján, így meghatároztam mely események számítanak kritikus eseménynek. Egy kritikus esemény egy olyan esemény, amely bekövetkezése esetén az aktuális szolgáltatási terv frissítése összetettebb algoritmust igényelhet, így újratervezésre lehet szükség. **A jelenlegi D(C)ARP megoldások közül egyik sem veszi figyelembe az összes lehetséges eseményt**, és csak egy olyan megoldás van, ami az összes kritikus eseményt figyelembe veszi (lásd D. függelék D.1.3. alszakasza).

Ahhoz, hogy a vizsgált probléma modellje a valós problémához minél jobban közelítsen, a változtatásokat a járművekről és az úthálózatról gyűjtött információk alapján kell elvégezni. Például, információkat szolgáltathatnak a járművek vezetői a végrehajtott feladatokról, a járművek GPS-e a járművek aktuális helyzetéről, és (közvetve) a közúti járőröző drónok az utak aktuális állapotáról [45]. **A jelenleg létező D(C)ARP megoldások mind azt feltételezik, hogy a járművek tökéletesen követik a kiadott szolgáltatási tervet, ami nem valósághű.** Lehetnek eltérések a terv és a tényleges végrehajtás között például olyan esetekben, amikor olyan nem megfigyelhető dinamikus esemény következik be, ami a járművezetőt arra kényszeríti, hogy más útvonalat kövessen. Ebből kifolyólag szükség van adat-vezérelt DCARP megoldásra.

Nemrégiben bemutattak egy keretrendszert, amely lehetővé teszi a statikus CARP megoldó algoritmusok használatát a DCARP példák szolgáltatási terveinek újratervezésére [46]. Ennek a keretrendszernek a használata hatékonyabbnak bizonyult, mint a meglévő megoldásoké, amelyek általában csak néhány fajta dinamikus esemény kezelésére képesek, és emellett lassú futási idővel is küzdenek. **Ezek a megoldások azonban csak az aktuális megoldás teljes újratervezését teszik lehetővé, ami időigényes lehet, és több jármű útvonaltervét is megváltoztathatja.** Van például egy olyan VRP-vel foglalkozó munka, amely egy jármű meghibásodása esetén csak egy jármű útvonaltervét módosítja (a meghibásodott jármű nem teljesített feladatainak kiszolgálása érdekében) és jó eredményeket tud elérni [47].

Célkitűzések

Az előző szakaszban kiemelt problémák megoldása érdekében a következő célokat tűztem ki:

- Egy ABC algoritmus kifejlesztése a CARP megoldására (**CARP-ABC algoritmus**), annak reményében, hogy a futási ideje és/vagy az általa adott megoldás minősége jobb lesz, mint a korábbi megoldó algoritmusoké.
- Egy **közepes lépésméretű mozgási művelet** kifejlesztése **a CARP-hoz**, ami a CARP-ABC algoritmus hatékonyságát növeli.
- Egy **adatvezérlet DCARP keretrendszer** kifejlesztése, amely
  - felhasználja a járművek tevékenységéről rendelkezésre álló adatokat (azaz adat-vezérelt), mivel előfordulhatnak eltérések az útvonaltervek és azok végrehajtása között;

– képes kezelni az összes lehetséges eseményt;
– csak akkor végez újratervezést, ha olyan kritikus esemény következik be, ami megvalósíthatatlanná és/vagy elavulttá teszi az aktuális szolgáltatási tervet;
– újratervezés esetén igyekszik megtalálni a legkevesebb költséggel járó (megvalósítható) szolgáltatási tervet, de emellett megpróbálja minimalizálni a módosított útvonaltervek számát.

Megoldások

A **CARP-ABC algoritmus**ból összesen két változatot készítettem: egy felfedezésre és egy feltárásra összpontosító változatot. A **felfedezésre összpontosító változat** [48, 49] célja, hogy minél változatosabb megoldásokat biztosítson, ezért azt statikus CARP megoldóként ajánlott használni. Ezzel szemben a **feltárásra összpontosító változat** [50, 51] célja, hogy minél hamarabb találjon minél jobb megoldást, ezért azt DCARP megoldóként javasolt alkalmazni.

Létrehoztam egy új, közepes lépésméretű mozgási műveletet, a **részútvonalterv műveletet** [49]. Ez a művelet egyszerre csak egy útvonalterv feladatainak sorrendjét és kiszolgálási irányát módosíthatja, így a szolgáltatási terv megvalósíthatósága biztos nem módosul. A többi művelet mellett ezt a műveletet is alkalmazzák a CARP-ABC algoritmusok.

Létrehoztam egy **adat-vezérelt DCARP keretrendszert** [52], ami minden lehetséges eseményt képes kezelni. A keretrendszer feladata, hogy a megfigyelt eseményeket feldolgozva aktualizálja a problémát (új DCARP példát generálva) és a szolgáltatási tervet. Egy kritikus esemény bekövetkezése esetén egy (D)CARP megoldó algoritmust hív segítségül a szolgáltatási terv újratervezéséhez. A keretrendszer részeként egy **minimális újratervező (RR1)** algoritmust [52] is létrehoztam, amely célja, hogy a megvalósíthatatlanná vált szolgáltatási tervet megvalósíthatóvá tegye, minimális számú útvonalterv módosítás mellett.

A DCARP megoldó algoritmusok valósághű környezetben való teszteléséhez a DCARP keretrendszerből két változatot is készítettem, amelyek leginkább abban különböznek, hogy hogyan kerülnek előállításra a szimulációhoz használt adatok:

- Az első változat (**az esemény generátort használó DCARP keretrendszer**) [52, 49] egy esemény létrehozó modult használ mind szolgáltatási folyamat végrehajtását leíró események (utazási vagy szolgáltatási események), mind a váratlanul bekövetkező, példát módosító események generálásához.

- A második változat (**a forgalom szimulációt használó DCARP keretrendszer**) [50, 51] valós úthálózatot és forgalmi adatokat felhasználó forgalom szimulációs szoftvert használ az utazási és szolgáltatási események generálásához, valamint a forgalom változást (forgalom növelést vagy csökkenést) leíró események létrehozásához.

1.3. A dolgozat felépítése

A dolgozat három fő fejezetre tagolódik, amelyek közül az 2., és a 3. fejezetek a két fő kutatási területet foglalják magukba, a dolgozat elején pedig egy általános bevezetés olvasható. A két fejezet azonos szakaszokra van osztva:

- **Irodalmi áttekintés**: Ez a szakasz a javasolt megoldásokhoz kapcsolódó szakirodalmat mutatja be és vizsgálja.

- **Előzetes fogalmak**: Ez a szakasz a javasolt megoldásokkal kapcsolatos alapfogalmakat mutatja be, hogy segítsen megérteni azok működését. A fogalmak csak röviden kerülnek bemutatásra, a részletesebb leírások a hivatkozott munkákban találhatók.

- **Javasolt megoldás**: Ez a szakasz egy javasolt megoldást mutat be részletesen. Minden megoldásnak külön szakasza van. A használt jelölések a „Jelölések" között megtalálhatók.
  - A 2. fejezetben itt az általam kifejlesztett előtag-igazítás alapú MOCC módszer (2.3. alszakasz), valamint a folyamatadatokat (az eredeti eseményadatokat és az igazítási adatokat) vizualizáló módszerek (2.4. alszakasz) kerülnek bemutatásra.
  - A 3. fejezetben itt a statikus és dinamikus kapacitáskorlátos ív útvonaltervezési problémához (CARP és DCARP) általam kifejlesztett megoldások kerülnek ismertetésre. Ezek között szerepel egy (D)CARP-hoz használható új mozgási művelet, a részútvonalterv művelet (3.3. alszakasz), két új (D)CARP megoldó algoritmus, a CARP-ABC algoritmusok (3.4. alszakasz), valamint az RR1 algoritmust alkalmazó adat-vezérelt DCARP keretrendszer (3.5. alszakasz), amihez a hatékonyság vizsgálatok elvégzéséhez két változatot is készítettem.

- **A javasolt megoldások hatékonyságának vizsgálata**: Ez a szakasz a fejezetben bemutatott megoldások hatékonyságának vizsgálatát és azok eredményeit mutatja be.

- **Összefoglalás**: Ez a szakasz a fejezetben bemutatott megoldások működését és a hozzájuk kapcsolódó hatékonyság vizsgálatok eredményeit foglalja össze.

A dolgozatban bemutatott megoldások által használt algoritmusok rendszerezése a B. függelékben olvasható.

2. fejezet

Több perspektívás online megfelelőség-ellenőrzés

2.1. Irodalmi áttekintés

Ez a szakasz a javasolt módszerekhez kapcsolódó szakirodalmat mutatja be és vizsgálja. Először a megfelelőség-ellenőrzési megoldásokhoz (azon belül a több perspektívás és online megoldásokhoz részletesebben), majd a folyamat vizualizációs módszerekhez nyújt áttekintést.

2.1.1. Megfelelőség-ellenőrzési megoldások

Az első megfelelőség-ellenőrzési megoldást 2005-ben Rozinat dolgozta ki, egy **token visszajátszás** (*token replay*) megfelelőség-ellenőrző algoritmus formájában, ami az eseménynapló egy nyomvonalát próbálja visszajátszani a folyamatmodellen [53]. A második mérföldkő 2009-ben volt, amikor Pesic és társai bemutatták az első **szabályellenőrzés**en (*rule checking*) alapuló megoldást, ami a folyamatmodellből levezetett szabályok teljesülését ellenőrzi az eseménynaplóban lévő nyomvonalakra vonatkozóan [54]. A megfelelőség-ellenőrzés következő mérföldköve 2011-ben volt, amikor Adriansyah bevezetett egy **igazítások**on (*alignments*) alapuló megfelelőség-ellenőrző algoritmust, ami az eseménynapló egy nyomvonalát a folyamatmodell egy végrehajtási szekvenciájához igazítja [55]. A másik két megközelítéssel ellentétben az igazítások képesek az eltérések és a megfelelőség eseményszintű kifejezésére. Napjainkban ezt tekintik az alap megfelelőség-ellenőrzési megoldásnak.

Az évek során számos más megfelelőség-ellenőrzési megoldást javasoltak, de a legtöbbjük a fent említett három megközelítés továbbfejlesztése. 2019-ben Dunzer és társai alapos irodalomfeldolgozást végeztek ebben a témában és számos következtetést vontak le. Többek között rámutattak a több perspektívás megfelelőség-ellenőrzési megoldások hiányára, kiemelve azok fontosságát [56]. 2022-ben Carmona és társai a [18] publikációban egy frissebb képet nyújtottak a szakirodalom jelenlegi helyzetéről.

A disszertációm keretében csak a több perspektívás megfelelőség-ellenőrzési megoldásokat – azon belül is az igazításon alapuló megoldásokat – (2.1.2. alszakasz) és az online megfelelőség-ellenőrzési megoldásokat (2.1.3. alszakasz) elemzem.

2.1.2. Több perspektívás megfelelőség-ellenőrzési megoldások

Az igazításon alapuló több perspektívás megfelelőség-ellenőrzési megoldások három fő csoportba oszthatók:

- **adat-orientált igazítások**: figyelembe veszi a folyamat kezdetén megadott vagy a folyamat bármelyik tevékenysége által generált adatokat.
- **erőforrás-tudatos igazítások**: figyelembe veszi a szerepek és az erőforrások közötti kapcsolatokat.
- **integrált megközelítések**: a vezérlésfolyam perspektíva mellett több perspektívát (pl. adat és adatvédelem) is figyelembe vesz.

Adat-orientált igazítások

Az első több perspektívás megfelelőség-ellenőrzési megoldásokat de Leoni és társai mutatták be [11, 12]. A vizsgált folyamat több perspektívából történő leírásához úgynevezett adat Petri-hálókat (2.2.2. alszakasz) vezettek be, ami napjainkban is a leggyakrabban használt folyamatmodell formátum több perspektívás megfelelőség-ellenőrzés esetén. Az első több perspektívás megfelelőség-ellenőrzési megoldás még mindig a vezérlésfolyam perspektívát tekintették a legfontosabb perspektívának az eltérések azonosításában, a többi perspektívát „másodrendű állampolgárként" kezelték. (Az igazítások létrehozásához csak a vezérlésfolyamot vették figyelembe, a többi perspektívában felelhető eltéréseket csak utána ellenőrizték.) Ezt a problémát Mannhardt és társai úgy oldották meg, hogy lehetővé tették a különböző perspektívák teljesen testreszabható módon történő kiegyensúlyozását [13, 57].

Felli és társai egy *megfelelőség kiszámítási moduló elméletek* (*Computing Conformance Modulo Theories*, röviden *CoCoMoT*) keretrendszert mutattak be, ami egy olyan átfogó megközelítést nyújt, amely a kielégíthetőségi modulelméletek (*Satisfiability Modulo Theories*, röviden *SMT*) elméletén és gyakorlatán alapul [58, 59, 60, 61, 62]. A CoCoMoT megközelítés legnagyobb előnye, hogy egy NP-ben futó megfelelőség-ellenőrzési eljárást eredményez, amely a probléma szempontjából optimális, szemben a korábbi, exponenciális időben futó megközelítésekkel ([11, 12, 13, 57]).

Gianola és társai egy közelítő módszert javasoltak a több perspektívás igazítás kiszámítására: ahelyett, hogy a naplóban rendelkezésre álló, a folyamat nyomvonalára vonatkozó teljes tudás alapján számítanák ki az optimális igazításokat, olyan veszteséges nyomvonal-kódolásokon alapuló közelítő igazításokat végeznek, amelyek csak bizonyos, a nyomvonalra vonatkozó információkat vesz figyelembe [63, 64].

Erőforrás-tudatos igazítások

Alizadeh és társai az adat Petri-hálók helyett úgynevezett „Létrehozás, olvasás, frissítés és törlés" (*Create Read Update Delete*, röviden *CRUD*) mátrixokat alkalmaznak [65]. A CRUD-mátrix az adatobjektumokat a folyamati logikához (azaz egy Petri-háló helyeihez vagy átmeneteihez) kapcsolja, így egy tevékenység csak akkor következik be, ha a kapcsolódó adatok lehetővé teszik vagy kikényszerítik azt. Ez a megoldás azáltal veszi figyelembe az erőforrás perspektíváját, hogy megvizsgálja az eseménynaplóban szereplő különböző adatműveleteket, és ellenőrzi, hogy ezeket a műveleteket engedélyezett erőforrások hajtották-e végre vagy sem.

Integrált megközelítések

Mozafari Mehr és társai egy olyan kiegyensúlyozott több perspektívás megfelelőség-ellenőrzési megoldást javasoltak, amely egyszerre veszi figyelembe a vezérlésfolyam, az adat és az adatvédelem perspektívákat, hogy átfogó betekintést nyújtson a folyamat működésébe [66, 67]. Később ezt a módszert kiegészítették kontextus helyekkel, amelyeket az igazítási eredmények magyarázatára és az eltérések kontextuális magyarázatára használnak [68, 69]. Ezenfelül, a gyakori szabálytalan viselkedési minták azonosításához egy olyan automatizált elemzési eljárást mutattak be, amely az említett megfelelőség-ellenőrzési módszer által észlelt eseti szintű eltéréseket veszi bemenetként, és a teljes folyamatvégrehajtás (eseménynapló szintű) teljes folyamatában feltárja a szabálytalan viselkedési mintákat, majd ezeket a mintákat szerepek, felhasználók és a rendszertől eltérő viselkedés szerint kategorizálja [70]. Az említett megoldásokat ProM folyamatbányászati eszközben használható pluginokként valósították meg [71, 72].

Egyéb megoldások

Zhang és társai az optimális igazítás kialakításakor a fuzzy halmazok elméletét használták az eltérések súlyosságának figyelembevételéhez (azaz a költségfüggvényben), hogy pontosabb diagnosztikát tudjanak generálni [73].

A több perspektívás megfelelőség-ellenőrzés esetében még mindig problémát jelenthet az eredmények számszerűsítése. Ezen problémák kezelése érdekében, Banham és társai bevezettek két új fogalmat az adattudatos megfelelőség-ellenőrzéshez; az őr-visszaidézést (*guard-recall*) és az őr-precízséget (*guard-precision*) [74].

2.1.3. Online megfelelőség-ellenőrzési megoldások

Az első olyan megoldást, amely képes az elvárt viselkedésektől való eltéréseket még a folyamat példány lefutása közben észlelni Weber és társai dolgozták ki 2015-ben [75]. Megközelítésüket CCaaS-ben (*Online Conformance Checking as a Service*, röviden *CCaaS*) valósították meg. Ez egy token visszajátszás alapú RESTful szolgáltatás, amely minden egyes esemény megfigyelését követően eldönti, hogy az illeszkedik-e vagy sem az adott BPMN folymatmodellre. A vezérlésfolyam perspektíván kívül az idő perspektívát is vizsgálja, azaz képes időzítési anomáliák észlelésére. Emiatt több perspektívás online megfelelőség-ellenőrzési megoldásnak is tekinthető, bár más fontos perspektívákat (pl.: adat vagy erőforrás) nem vizsgál. Ez a megközelítés azonban nem biztosít idő- és memóriakorlátokat, ezért számítási szempontból nem tekinthető hatékonynak. Ezt a problémát orvosolta Burattin és Carmona a [8] publikációban bemutatott és a [76] publikációban implementált online megfelelőség-ellenőrzési keretrendszerben. A folyamatlefutás megfelelőségének vizsgálatához a nem megfelelő tüzelési szekvenciákat hozzáadták a folyamat-modellben szereplő szabályos tüzelési szekvenciákhoz, a régiók elméletének felhasználásával [77]. Így ha egy nyomvonal nem megfelelő tüzelési szekvenciát követ, akkor azt nem megfelelőnek minősíti. Bár ez a módszer számítási idő szempontjából hatékonyabbnak tekinthető a [75] publikációban bemutatottnál, csak a vezérlésfolyam perspektívát vizsgálja. Továbbá, ez a módszer is csak azt jelzi, hogy a nyomvonal megfelel-e a modellnek vagy sem, további információt nem szolgáltat.

Az azóta bemutatott online megfelelőség-ellenőrzési megoldások többsége általánosítva három fő megközelítés típusra oszthatók:

- metrika alapú megközelítések (pl.: [9] és [10]),
- előtag-igazítás alapú megközelítések (pl.: [6]), illetve
- metrika és előtag-igazítás alapú megközelítések (pl.: [78]).

A metrika alapú megközelítések gyors számítást és (az adott mutatókkal) átfogó nézetet nyújtanak, de nem jelzik egyértelműen az eltérő tevékenységeket. Emiatt csak riasztási célokra alkalmasak. Az előtag-igazítás alapú megközelítések jól láthatóan jelölik a váratlan és a kihagyott tevékenységeket, de általában hosszabb számítási időt igényelnek. Emellett nem képesek kezelni a meleg indításos helyzeteket (azaz, amikor a megfigyelés később kezdődik, mint a folyamatlefutás), illetve nem képesek magyarázni az igazítás költségének a megbízhatóságát.

Metrika alapú megközelítések

A [8] publikációban bemutatott és a [9] publikációban továbbfejlesztett viselkedési mintákon alapuló módszer a megfigyelt események tevékenységéből képzett párok (azaz viselkedési minták) a kapcsolódó folyamatmodellt alkotó viselkedési mintákkal való megfelelőségét vizsgálja. Ez a megoldás a megfelelőségen kívül két más mutatót is alkalmaz: a teljességet és a megbízhatóságot. A teljesség azt mutatja meg, hogy a teljes nyomvonal a kezdettől lett-e megfigyelve, így ez a mutató a melegindításos helyzetek esetében hasznos. A megbízhatóság annak a valószínűségét jelzi, hogy a megfelelőségi pontszám változatlan marad-e a folyamatlefutás végéig.

A [10] publikációban egy rejtett Markov-modell (*Hidden Markov Model*, röviden HMM) alapú megközelítést vezettek be, ami először megkeresi az esetek nyomvonalát a kapcsolódó folyamatmodellben, majd értékeli azok megfelelőségét. Ez a megoldás a megfelelőségen kívül csak a teljesség mutatót használja.

Előtag-igazítás alapú megközelítések

Az offline használt igazítások feltételezik, hogy a folyamat példányok lefutása befejeződött, azaz a nyomvonaluk teljes és már nem fog változni. A (még) nem befejezett folyamat példányok lefutásának a nyomvonalát (azaz az előtagokat) hibásnak tekintik, így nem használhatók online környezetben. A [79] publikációban bemutatott, majd a [6] publikációban az online környezetre adaptált előtag-igazítások enyhítik ezt a problémát azáltal, hogy nem büntetik azokat a nyomvonalakat, amelyek nem a modell szerint fejeződnek be. Emellett optimalizálásra kerültek a számításigényes előtag-igazítások egy könnyű, modell-szemantika alapú előtag-igazítási szintézis módszer bevezetésével.

A [80] publikációban javasolt módszer az online előtag-igazítások inkrementális kiszámításával pontos megfelelőség-ellenőrzést hajt végre. Az új események megfigyelésekor inkrementálisan bővíti a szinkron szorzathálót a legrövidebb útvonal megkeresése érdekében, így megtalálva az optimális előtag-igazítást.

Az előtag-igazítással kapcsolatos munkák egy része a számítás memóriaigényének javítására törekedik [7, 81, 82, 83]. Raun és társai a [84] publikációban és a [85] publikációban egy trie (más néven digitális fa vagy előtag fa) adatszerkezeten alapuló közelítő algoritmust mutattak be az előtag-igazítások kiszámítására, amely felülmúlja a korábbi módszereket. Ezt követően, a [82] publikációra és a [84] publikációra építve, a [78] publikációban egy olyan módszert mutattak be, amely

előtag-igazításokat ad eredményül, miközben kezeli a melegindítási helyzeteket, és kiszámítja a megbízhatóságot és a teljességet is.

A memória korlátozás egyik kezelési lehetősége, hogy csak korlátozott számú aktív esetet tartunk meg a memóriában. Ennek megfelelően a legrégebben frissített esetek feledésbe merülnek, feltételezve, hogy azok már nagy valószínűséggel befejeződtek. Ez hiányzó előtag problémához ([86]) vezet abban az esetben, ha egy még nem bejeződött, de elfelejtett esethez tartozó új eseményt (azaz egy árva eseményt) figyelünk meg. Az OCC-technika az árva eseményt egy új esethez tartozónak fogja nézni és emiatt (tévesen) nem megfelelőnek nyilvánítva az esetet az elfelejtett előtagja miatt. Annak érdekében, hogy hatékonyan csökkentve legyen a memóriában megőrzendő eseményadatok mennyisége anélkül, hogy a hiányzó előtag probléma jelentkezne, Zaman és társai többféle megközelítést mutattak be a [81] publikációban és a [83] publikációban. A [81] publikációban három inkrementális megközelítést fogalmaztak meg, amiknek az a lényege, hogy a meghatározott határérték felett lévő állapotokat elfelejtik, és azoknak csak az értelmes összefoglalását tartják meg. A [83] publikációban kétféle megközelítést mutatnak be: állapotfüggő és állapotmentes. Mindkét megközelítés csak meghatározott számú esetet tart meg a memóriában, és túlcsordulási helyzetben a kijelölt felejtési kritériumok segítségével megfontoltan elfelejti a felesleges eseteket. Hasonlóan a [81] publikációban bemutattakhoz, az állapotfüggő megközelítés, amikor elfelejt egy esetet, csak az értelmes összefoglalását tartja meg. Ez tartalmazza az eset aktuális pozícióját a folyamatban, így új esemény megfigyelése esetén onnan tudja folytatni az előtag-igazítás számítását. Az állapotmentes megközelítés nem tart meg semmilyen információt az elfeledett esetekről. Új esemény megfigyelése esetén, gépi tanulási osztályozási technikák segítségével becsüli meg az eset aktuális pozícióját a folyamatban, majd onnan folytatja az előtag-igazítás számolását.

Egyéb

Az eddig bemutatott online megfelelőség-ellenőrzési megoldások előíró folyamatmodellt feltételeztek. Ezekkel ellentétben, a [87] publikációban egy olyan online lágy megfelelőség-ellenőrzési megoldás került bemutatásra, ami leíró folyamatmodellt feltételez. Ebből adódóan, a megfelelőség bármilyen perspektívából kiszámítható, nem csak a vezérlési folyamból. Viszont egyszerre csak egy perspektívát tud vizsgálni.

A [88] publikációban Stertz és társai olyan igazítás költségszámítási módszert mutattak be, amely lehetővé teszi az eltérésenkénti költségek eltérésenkénti korrekcióját online és offline is. Ez a megoldás többperspektívásnak mondható, mivel az események adatelemeit is vizsgálja. Viszont az optimális igazítás az alap költségfüggvény alapján kerül meghatározásra, ami csak a vezérlésfolyamot veszi figyelembe. Ezt követően kerül az igazítás költsége módosításra egy speciális költségfüggvény segítségével, ha a szinkron lépések esetén eltérés van az elvárt és a megfigyelt adatérték(ek) között.

2.1.4. Folyamat vizualizációs megoldások

Ebben az alszakaszban áttekintést nyújtok a meglévő folyamatbányászati szoftvereszközökhöz, majd elemzem az általuk használt vizualizációs módszereket az eseményadatokhoz és a megfelelőség-ellenőrző algoritmusok kimeneteihez, ideértve az igazításokat is.

Folyamatbányászati eszközök

Jelenleg számos különböző folyamatbányászati szoftvereszköz áll rendelkezésre. A [89] publikációban a négy legnépszerűbb eszköz teljesítményét és jellemzőit hasonlították össze, amelyek a következők: ProM, Disco, Celonis és My-Invenio. A [90] publikációban 27 elérhető eszközt sorolnak fel, de részletes vizsgálatot csak az előbb említett négy eszközön és az Apromore-on végeztek. A legkiterjedtebb elemzés a [91] publikációban található, ahol 17 eszközt vizsgáltak meg alaposan. Emellett egy olyan honlapot is létrehoztak, amely a létező folyamatbányászati szoftvereszközök folyamatosan frissített listáját tartalmazza, ami egyúttal platformot nyújt azok összehasonlító elemzéshez.

A [91] publikáció megállapításai szerint a vizsgált eszközök korlátozottan képesek az időbélyegek kezelésére. Egy eszköz csak olyan eseményadatokat tud kezelni, amik pontosan egy, vagy amik pontosan kettő időbélyeggel rendelkeznek (azaz a két lehetőség közül csak az egyiket tudja megfelelően kezelni). A folyamatfelfedezés vonatkozásában egyes eszközök képesek megjeleníteni a folyamatok végrehajtási változatait (azaz nyomvonalváltozatokat (*trace variant*)), ezáltal részletes képet nyújtva az egyes esetekről. Bár ezek hasznosak a megfigyelt folyamatvégrehajtások körének vizualizálásához, nem nyújtanak kontextuális információt. Például, nem szemléltetik a folyamatban lévő esetek számát, az adott időpontban végzett tevékenységeket, az érintett erőforrásokat vagy azok helyét. A megfelelőség-ellenőrzés esetében egyes eszközök képesek a folyamatmodellre rávetített, összesített képet adni az eltérésekről. Ezen túlmenően képesek felsorolni az eltéréseket (a nem kívánt és a hiányzó tevékenységeket), valamint a nem megfelelő tevékenység sorozatokat. Ezek a megjelenítések azonban gyakran szöveges vagy táblázatos formátumra korlátozódnak.

PM4Py-ra épülő eszközök

A [91] publikációban vizsgált eszközök elsődleges korlátja, hogy az Apromore és a ProM kivételével nem nyílt forráskódúak. Ez korlátozza az új funkciók beemelésének a lehetőségét. Ennek következtében számos fejlettebb és legújabb folyamatbányászati algoritmus nincs beépítve ezekbe az eszközökbe. Ezzel szemben a PM4Py [92] nyílt forráskódú folyamatbányászati programkönyvtárat kínál Python nyelvre, amely könnyű bővíthetőséggel rendelkezik. Ezt a könyvtárat alapul véve számos folyamatbányászati alkalmazást fejlesztettek ki.

A [93] publikáció egy olyan megközelítést mutat be, amely az eseménynaplók elemzésével és az eredmények vizuális bemutatásával azonosítja az üzleti folyamatok szűk keresztmetszeteit (*bottleneck*). Ez a módszer a tevékenységek és a nyomvonalak időtartamát képes az idő függvényében pontdiagramok formájában ábrázolni. Ennek a megoldásnak az a hiányossága, hogy az egyes nyomvonalakra vonatkozó információk helyett az összegzett értékekre helyezi a hangsúlyt.

A [94] publikáció a PMTK nevű folyamatbányászati eszközkészletet mutatja be, amely a ProM-hoz [95] hasonlóan képes a nyomvonalváltozatok és az események időbeli megjelenítésére egy pontozott diagram segítségével. Ugyanakkor nem képes megfelelőség-ellenőrzést végezni.

A [96] publikáció egy Smyrida nevű moduláris szoftverrendszert mutat be, amely szintén képes a nyomvonalváltozatok és azok gyakoriságának megjelenítésére, de nem tudja azokat idővonalon megjeleníteni. A rendszer képes megfelelőség-ellenőrzést végezni, de az igazításokat csak táblázatokban jeleníti meg.

A [97] publikáció a Cortado nevű interaktív folyamatfelfedező eszközt mutatja be. Bár képes megfelelőség-ellenőrzést végezni és az igazításokat vizualizálni, az igazítási mozgások megjelenítésében korlátozva van. Konkrétan, amikor egy igazítást egy nyomvonalra (vagy variánsra) vetít, a modell mozgások nem jelennek meg. Ugyanez fordítva is igaz. Ha egy igazítást egy folyamatmodellre vetít, az eseménynapló mozgások nem jelennek meg. Ezenkívül, bár a statisztikai mérőszámokat, például a kiszolgálási idő és a várakozási idő hosszát (pl. minimum, maximum, átlag stb.) kivetíti a nyomvonalváltozatokra és a modellre, nincs lehetőség arra, hogy ezeket a várt értékekkel összevesse.

Vizualizációs módszerek eseményadatokhoz

Az eseménysorozatok online megjelenítéséhez használt vizualizációs megoldások közül kizárólag az idővonal-alapú vizualizációs megoldásokra fókuszáltam. Az ilyen vizualizációk az eseménysorozatokat grafikusan jelenítik meg, egy időbeli tengelyen igazítva őket, hogy az eseménysorozaton belüli események sorrendjét ábrázolják [98].

Az eseménysorozat adatok esetében az idővonal-alapú megjelenítések az idő ábrázolásmódja alapján kategorizálhatók:

- fix idővonal-alapúak (mint például a pontozott diagramok [95]),
- időtartam-alapúak (beleértve a Gantt-diagramokat, oszlopdiagramokat és azok változatait [99]), valamint
- a kettő kombinációja (amint azt az egyik munkámban bemutattam [17]).

Az egyes események ábrázolása általában az időbélyegük és egy másik kiválasztott attribútumuk, gyakran az esetazonosítójuk vagy az erőforrásuk alapján történik. Ezek az események grafikus objektumok (pl. körök vagy téglalapok) formájában jelennek meg, és a tevékenység alapján vannak színkódolva.

A jelenlegi vizualizációs megoldások azonban egy jelentős problémával küszködnek: az esemény átfedés problémájával. Ha több esemény azonos időbeli adatokkal rendelkezik, akkor a vizualizációban az egyik el fogja fedni a többit. Továbbá, a fix idővonalon alapuló vizualizációkban, amikor egy tevékenység végrehajtása több eseményt is átfog (például egy kezdő eseményt egy befejező esemény követ), ezek az események nincsenek egymással összekapcsolva. Ez a hiányosság nem érzékelteti a tevékenység végrehajtásának időtartamát.

Vizualizációs módszerek igazítási adatokhoz

A megfelelőség-ellenőrzési megoldások kimeneteinek – ideértve az igazítási adatokat is – vizualizálására jelenleg nem léteznek idővonal-alapú módszerek. Emiatt az összes fellelhető módszert megvizsgáltam.

A [100] publikációban Rehse és társai alaposan megvizsgálák, hogy a különböző folyamatbányászati eszközök hogyan jelenítik meg a megfelelőség-ellenőrzési eredményeket. A „Miért?” kérdéssel kapcsolatban az eszközök felhasználói felületein belül a vizualizációk felépítését vizsgálva négy fő csoportba sorolták őket:

1. a megfelelőség számszerűsítése,
2. a megfelelőség lebontása és összehasonlítása,
3. az eltérések lokalizálása és kimutatása,
4. az eltérések magyarázata és diagnosztizálása.

Én elsősorban a 3. kategóriára (az eltérések lokalizálása és kimutatása) koncentráltam, amely olyan módszereket tartalmaz, amelyek meghatározott folyamatválto-

zatokhoz mutatnak be igazításokat. Ezek a módszerek túlnyomórészt színkódolt folyamatábrákat és Chevron-diagramokat (színes nyílhegyek sorozatait) használnak a nyomvonalon belüli összes tevékenység igazításának ábrázolására.

Chevron-diagramokon alapuló vizualizációs módszerek léteznek a vezérlésfolyam igazításokhoz [55] és a több perspektívás igazításokhoz [101] is. Mindkét módszer színes Chevron-diagramokat használ az igazítási mozgások sorrendjének ábrázolásához, ahol a nyílhegyek színei a mozgások típusait jelölik. A több perspektívás igazításoknál van egy olyan változat is, ahol a nyílhegyek a tevékenységnek megfelelően vannak színezve, és a mozgástípusok pedig a nyilak feletti színes sávok formájában vannak ábrázolva.

2.2. Előzetes fogalmak

Ez a szakasz a javasolt módszerrel kapcsolatos alapfogalmakat mutatja be, hogy segítsen megérteni a működését. A fogalmak csak röviden kerülnek bemutatásra, a részletesebb leírások a hivatkozott munkákban találhatók.

Ebben a szakaszban először a javasolt MOCC módszer bemeneteihez kapcsolódó fogalmak kerülnek bemutatásra: a valós viselkedést leíró, rögzített események (2.2.1. alszakasz) és az elvárt viselkedést definiáló adat Petri-háló folyamatmodell (2.2.2. alszakasz). Ezt követően ismertetésre kerül a módszer kimenete, az igazítások (2.2.3. alszakasz), valamint az ezek előállításához kapcsolódó fogalmak (2.2.4-2.2.6. alszakaszok). A használt jelölések a „Jelölések" között megtalálhatók.

Ebben a szakaszban saját hozzájárulásaim a következők:

- A 2.2.1. alszakaszban az „egyszerű eseményfolyam" és az „összetett eseményfolyam" fogalmának bevezetése.
- A 2.2.4. alszakaszban az „optimális változó hozzárendelési probléma" (*Optimal Variable Assignment Problem*, röviden *OVAP*) elnevezése (a probléma maga nem új), valamint az OVAP gyorsítótár fogalmának kidolgozása.
- A 2.2.6. alszakaszban bemutatott inkrementális A* algoritmus módosítások.

2.2.1. Esemény és eseményfolyam

A legtöbb folyamatbányászati algoritmus a megfigyelt viselkedéseket tartalmazó bemenetként egy **eseménynaplót** (*event log*) vár, amely véges számú **esemény**t (*event*) tartalmaz megfigyelési sorrendben. Minden eseménynek rendelkeznie kell egy **esetazonosító**val (*case identifier*) és egy **tevékenység**gel (*activity*). Az esetazonosító azonosítja azt a folyamatpéldányt (vagy kontextust), amelyben a tevékenységet végrehajtották. Egy folyamat végrehajtása során olyan tevékenységek is végrehajtódhatnak, amik nem megfigyelhetők (nem tudjuk, vagy csak nem akarjuk megfigyelni) és ezért nem kerülnek rögzítésre az eseménynaplóban. Ennek megfelelően egy eseménynapló csak olyan eseményeket tartalmazhat, amik megfigyelhető tevékenységgel rendelkeznek. A tevékenységeknek vagy a neve, vagy az azonosítója kerül rögzítésre.

A valós események további információkat is tartalmazhatnak a végrehajtott tevékenységről, például a tevékenységet végrehajtó személyről vagy gépről (erőforrás)

vagy a tevékenység végrehajtásának időpontjáról (időbélyeg). Ezeket **esemény attribútum**oknak (*event attribute*) nevezzük.

Egy tevékenység végrehajtása, amelyet gyakran **tevékenységpéldány**nak (*activity instance*) is neveznek, több eseményből is állhat. Ezen események mindegyike a tevékenység különböző életciklus-átmeneteit írja le. A lehetséges átmenetek (másnéven tranzakciótípusok) átfogó listája megtalálható a [102] publikációban. A leggyakoribb átmenettípusok a *start* és a *complete*, amelyek a tevékenységpéldány végrehajtásának kezdetét, illetve végét jelölik. Ha ugyanaz a tevékenység többször is előfordulhat ugyanabban a kontextusban (azaz ugyanazzal az esetazonosítóval), akkor az eseménynek tartalmaznia kell a tevékenységpéldány azonosítóját is.

Online környezetben eseménynapló helyett **eseményfolyam**ot (*event stream*) feltételezünk. Az eseményfolyam **megfigyelhető egységek**ből áll. Egy megfigyelhető egység általában egy eseménynek felel meg, ezért az eseményfolyamot események (végtelen) sorozataként definiáljuk, ahol az esemény egy esetazonosítót és egy megfigyelhető tevékenységet tartalmazó rendezett pár [7]. Ha a kiegészítő információkat is bevonjuk, akkor az esemény egy rendezett hármas, amiben a harmadik elem az n darab esemény attribútum értékeit tartalmazó rendezett n-es.

1. Definíció (Esemény; Eseményfolyam)**.** Legyen $\mathcal{C}$ az esetazonosítók univerzuma, $\mathcal{A}$ a megfigyelhető tevékenységek univerzuma és $\mathcal{U}$ az értékek univerzuma. Legyen $\mathcal{U}^n$ n darab esemény attribútum értékeinek univerzuma oly módon, hogy $\mathcal{U}^n = \prod_{i=1}^{n} \mathcal{U}_i$ egy Descartes-szorzat, ahol $\mathcal{U}_i \in \mathbb{P}(\mathcal{U})$ az i-edik esemény attribútum értékeinek univerzuma az n darab esemény attribútum közül. Így $u^n \in \mathcal{U}^n$ egy rendezett n-es, ahol $u^n = (u_1, u_2, \ldots, u_n) \in \mathcal{U}_1 \times \mathcal{U}_2 \times \cdots \times \mathcal{U}_n$. Legyen $\mathcal{E} = \mathcal{C} \times \mathcal{A} \times \mathcal{U}^n$ a lehetséges események univerzuma. Egy $e = (c, a, u^n) \in \mathcal{E}$ esemény egy olyan rendezett hármas, ami az $a \in \mathcal{A}$ tevékenység végrehajtását írja le $u^n = (u_1, u_2, \ldots, u_n) \in \mathcal{U}^n$ attribútum értékekkel, egy $c \in \mathcal{C}$ esetazonosító által azonosított folyamatpéldány kontextusában. Egy E eseményfolyam az események végtelen sorozata, azaz $E \in \mathcal{E}^\star$.

A fenti definíció a [7] publikációban szereplő vezérlésfolyam (azaz esemény attribútumok nélküli) esemény és eseményfolyam definícióján alapul.

2.1. táblázat. A megfigyelhető egységre vonatkozó tartalmi követelmények a két eseményfolyam-típus esetében

Attribútum neve	**Egyszerű eseményfolyam**	**Összetett eseményfolyam**
esetazonosító	kötelező	kötelező
tevékenység	kötelező	kötelező
tevékenységpéldány azonosítója	kötelező	nem releváns
tranzakciótípus	kötelező	nem releváns
időbélyeg	1 db kötelező	2 db kötelező (kezdő és befejező)
egyéb attribútumok (pl. erőforrás)	opcionális	opcionális

Ebben a munkában az eseményfolyamokat két különböző típusba soroltam az alapján, hogy az egyes megfigyelhető egységek hány eseménynek felelnek meg: egyszerű eseményfolyam és összetett eseményfolyam. Az **egyszerű eseményfolyam**ban

minden megfigyelhető egység egy adott tevékenységpéldányhoz tartozó, egyedi eseményről tartalmaz információt. Ezzel szemben az **összetett eseményfolyam**ban minden megfigyelhető egység egy tevékenységpéldány összes eseményének adatait tartalmazza, amelyek részben aggregált, részben pedig tömörített formában jelennek meg. Az események időbélyegei közül például csak a kezdő és a befejező időpont szerepel, míg az események más tulajdonságai összesítve kerülhetnek tárolásra. A 2.1. táblázat összefoglalja azokat az attribútumokat, amelyek értékei az egyes eseményfolyam-típusok megfigyelhető egységein belül jelen vannak. A megfigyelhető egységek további attribútumokat is tartalmazhatnak. Az attribútumértékek sorrendje egy megfigyelhető egységen belül rugalmas, de végig következetesnek kell maradnia.

Eseménynapló lépés és nyomvonal

Az azonos esetazonosítót tartalmazó eseményekből felépíthető az adott folyamatpéldány **nyomvonal**a (*trace*). Az eseményadatok egyszerű feldolgozása érdekében a megfigyelt események külön-külön kerülnek naplózásra a nyomvonalakban, az esetazonosítójuk értéke alapján, de az esetazonosítójuk nélkül. Ezeket **eseménynapló lépés**eknek (*log step*) nevezzük.

2. Definíció (Eseménynapló lépés; Nyomvonal)**.** Legyen $S_l \subseteq \mathcal{A} \times \mathcal{U}^n$ a lehetséges eseménynapló lépések halmaza. Így egy $s_l = (a, u^n) \in S_l$ eseménynapló lépés egy olyan rendezett pár, amely egy $a \in \mathcal{A}$ tevékenység és $u^n \in \mathcal{U}^n$ attribútumértékekből áll. A $\sigma = \langle s_{l_1}, s_{l_2}, \ldots \rangle$ nyomvonal egy adott $c \in \mathcal{C}$ esetazonosítót tartalmazó megfigyelt eseményekből generált eseménynapló lépések véges sorozata, azaz $\sigma \in S_l^\star$.

Egy rendezett n-es (pl. u^n esemény attribútum értékek vagy e esemény) vagy sorozat (pl. σ nyomvonal) i-edik elemének a kinyeréséhez a π_i projekciós függvényt használjuk. Például, kinyerhetjük a harmadik attribútum értékét az u^n attribútum értékekből a következő képpen: $\pi_3(u^n) = u_3$.

2.2.2. Adat Petri-háló

Az **adat Petri-háló** (*Data Petri Net*, röviden *DPN*) a Petri-hálók speciális típusa, amelyet a vezérlésfolyam perspektíva és a többi perspektíva kölcsönhatásainak megragadására vezettek be a [103] publikációban, majd később a [104] publikációban újragondoltak.

A DPN az adatértékeket globálisan definiált **folyamatváltozó**kban (*process variable*) tárolja, amelyeket meghatározott átmenetek **írási művelet**eivel (*write operation*) lehet frissíteni. Az új adatértékek az átmenet tüzelése előtt ideiglenesen **prímváltozó**kban (*prime variable*) kerülnek eltárolásra és csak az átmenet tüzelését követően kerülnek a változókba. A DPN-ben az átmeneteknek lehetnek **adatfüggő őr**ei (*data-dependent guard*). Egy őrrel rendelkező átmenet csak akkor tüzelhet, ha az őr kielégítésre kerül. Ez azt jelenti, hogy az őrhöz rendelt **őrkifejezés** (*guard expression*) teljesül, azaz logikai igaz értéket ad vissza a kiértékelése során. Az őrkifejezés bármilyen logikai és relációs operátorokból felépített lineáris képlet lehet, amiben különböző változók és prímváltozók is szerepelhetnek. A DPN tartalmazhat **láthatatlan átmenet**eket (*invisible transition*) is, amelyek nem megfigyelhető eseményekhez köthetők.

3. Definíció (Változók; Prímváltozók; Változók értékhalmaza)**.** Legyen $\mathcal{U}$ az értékek univerzuma. Az összes változó (elnevezésének) véges halmazát V-vel jelöljük. A prímváltozók véges halmazát $V' = \{v' \mid v \in V\}$-vel jelöljük. Minden egyes $v \in V$ változóhoz egy $v' \in V'$ prímváltozót párosítunk, azaz

$$v \leftrightarrow v', \quad \forall v \in V,\, v' \in V' \tag{2.1}$$

A $val : V \to \mathbb{P}(\mathcal{U})$ függvénnyel definiáljuk a változók (potenciálisan) végtelen elemszámú értékhalmazát. Tehát, ha adott egy $v \in V$ változó (vagy egy $v' \in V'$ prímváltozó, aminek v a párja), a $val(v)$ függvényt használjuk az adott változó által felvehető értékek halmazának kinyerésére.

4. Definíció (Változó hozzárendelések)**.** Legyen $\mathcal{U}$ az értékek univerzuma, V a változók halmaza és $val : V \to \mathbb{P}(\mathcal{U})$ a változók értékhalmazát definiáló függvény. Egy $w \in V \nrightarrow \mathcal{U}$ változó hozzárendelés egy olyan parciális függvény, ami $v \in V$ változókhoz rendel $u \in \mathcal{U}$ értékeket. A w függvény értelmezési tartományát $dom(w)$-vel jelöljük. A w változó hozzárendelés akkor érvényes, ha minden $v \in dom(w)$ esetén $w(v) \in val(v)$. Ebből adódóan, az összes lehetséges érvényes változó hozzárendelés halmaza a $W = \{w \in V \nrightarrow \mathcal{U} \mid \forall_{v \in dom(w)}(w(v) \in val(v))\}$ halmazzal definiálható.

5. Definíció (Őrkifejezés)**.** Legyen V a változók halmaza és V' a prímváltozók halmaza. Jelöljük a $v \in V$ változók és $v' \in V'$ prímváltozók feletti összes logikai kifejezés univerzumát $EXPR(V \cup V')$-vel. Egy lineáris őrkifejezés egy olyan $expr \in EXPR(V \cup V')$ logikai formula, amely igazra vagy hamisra értékelődik ki.

6. Definíció (Őrkifejezés kiértékelő függvény)**.** Legyen $\mathcal{U}$ az értékek univerzuma. Legyen V a változók halmaza, V' a prímváltozók halmaza és $val : V \to \mathbb{P}(\mathcal{U})$ a változók értékhalmazát definiáló függvény. Legyen $expr \in EXPR(V \cup V')$ egy őrkifejezés. Az őrkifejezés igazságértékét az $eval_{V,val} : (EXPR(V \cup V') \times W) \to \{\texttt{true}, \texttt{false}\}$ őrkifejezés kiértékelő függvénnyel értékeljük ki. Az $eval_{V,val}$ függvény a $w \in W$ változó hozzárendelés esetén igazra vagy hamisra értékeli a kifejezést. Ha a kifejezés kiértékeléséhez szükséges egy vagy több változó nem definiált (azaz $v \notin dom(w)$), akkor `false` (azaz hamis) értéket ad vissza.

7. Definíció (Adat Petri-háló)**.** Az adat Petri-hálót az $N = (P, T, F, V, val, in, wr, gd)$ rendezett nyolcassal definiáljuk, ahol

- P a helyek véges halmaza,
- T az átmenetek véges halmaza,
- $F \subseteq (P \times T) \cup (T \times P)$ irányított élek véges halmaza, ami a helyek és az átmenetek közötti kapcsolatot írja le,
- V a folyamatváltozók (elnevezéseinek) véges halmaza,
- $val : V \to \mathbb{P}(\mathcal{U})$ egy $v \in V$ folyamatváltozó (potenciálisan) végtelen elemszámú értékhalmazát visszaadó függvény,
- $in : V \to \mathcal{U}$ egy $v \in V$ folyamatváltozó kezdeti értékét visszaadó függvény,
- $wr : T \to \mathbb{P}(V)$ egy $t \in T$ átmenethez előírt változóírási műveletek által érintett folyamatváltozók halmazát visszaadó függvény, és
- $gd : T \to EXPR(V \cup V')$ őrfüggvény, ami egy $t \in T$ átmenethez tartozó ($v \in V$ folyamatváltozókkal és $v' \in V'$ prímváltozókkal definiált) logikai őrkifejezést visszaadó függvény. Ha t átmenethez nem tartozik őrkifejezés, akkor $gd(t) =$ `true`.

A DPN állapota a Petri-háló jelölésének és az aktuális változó hozzárendelésének kombinációja.

8. Definíció (Adat Petri-háló állapota)**.** Legyen az $N = (P, T, F, V, val, in, wr, gd)$ egy adat Petri-háló. Az N lehetséges állapotainak halmazát az olyan (M, α) párok képezik, ahol

- $M \in \mathbb{B}(P)$ a (P, T, F) Petri-háló jelölése, és
- $\alpha \in \mathcal{U}^{|V|}$ a jelenlegi folyamatváltozó hozzárendelés.

A megfelelőség-ellenőrzés elvégzéséhez **címkézett adat Petri-háló** folyamatmodellt használunk. Ez egy olyan adat Petri-háló, ami címkefüggvényekkel segíti a modell eseményadatokkal történő összevetését. Az átmenetek és a megfigyelhető tevékenységek megfeleltetésére egy **tevékenységcímke függvény**t (*activity label function*) használunk, a folyamatváltozók és az esemény attribútumok megfeleltetésére pedig egy **változócímke függvény**t (*variable label function*). Feltételezve, hogy az eseményadatok egy eseményfolyamból származnak, a változócímke függvényben minden változót ahhoz a pozícióhoz rendeljük hozzá, ahol a megfelelő attribútum értéke található egy esemény attribútum értékeinek rendezett n-esén belül.

Feltételezzük, hogy a címkézett adat Petri-háló egyben **munkafolyamat-háló** (*workflow net*) is, tehát egy egyedi forrás és nyelő hellyel rendelkezik, valamint minden hely és átmenet a forrás és a nyelő közötti útvonalon van.

9. Definíció (Címkézett adat Petri-háló)**.** A címkézett adat Petri-hálót az $N_\lambda = (P, T, F, V, val, in, wr, gd, [p_i], [p_f], \lambda_a, \lambda_v)$ rendezett tizenkettessel definiáljuk, ahol az első nyolc elem az adat Petri-hálót definiálja. A további elemek a következők:

- $[p_i]$ kezdeti állapot jelölés egyedi $p_i \in P$ forrás hellyel,
- $[p_f]$ végső állapot jelölés egyedi $p_f \in P$ nyelő hellyel,
- $\lambda_a : T \to \mathcal{A} \cup \{\tau\}$ tevékenységcímke függvény, ami minden $t \in T$ átmenethez visszaadja a megfelelő tevékenységet, ha a tevékenység megfigyelhető (azaz $a \in \mathcal{A}$), különben τ-t ad vissza, és
- $\lambda_v : V \to \mathbb{N}^+_{\leq n}$ változócímke függvény, ami minden $v \in V$ folyamatváltozóhoz visszaadja a megfelelő esemény attribútum $0 < i \leq n$ index értéket, ami az u_i értéket azonosítja u^n-en belül.

Az adat Petri-hálókkal kapcsolatos pontos definíciók az [57] publikációban találhatók, példákkal együtt. Nagyobb különbség a λ_v változócímke függvény definiálásában van, mivel az [57] publikációban bemutatott megoldás az eseményeket eseménynapló formájában várja, a dolgozatomban bemutatott megoldás pedig eseményfolyam formájában.

Modell lépés és tüzelési szekvencia

Az érvényes, végrehajtott tüzeléseket **modell lépés**nek (*model step*) nevezzük. Egy modell lépés a tüzelt átmenetet és az átmenet által írt változó értékeket foglalja magába. Az egymást követő modell lépések sorozatát **tüzelési szekvenciá**nak (*firing sequence*) hívjuk.

10. Definíció (Modell lépés; Tüzelési szekvencia)**.** Legyen $S_m \subset T \times W$ az érvényes modell lépések halmaza. Így egy $s_m = (t, w) \in S_m$ (érvényes) modell lépés, amely a tüzelt $t \in T$ átmenetből és a $w \in W$ új változó hozzárendelésekből áll. A $\rho = \langle s_{m_1}, s_{m_2}, \ldots \rangle$ tüzelési szekvencia egy N (vagy N_λ címkézett) adat Petri-hálón visszajátszható modell lépések véges sorozata, azaz $\rho \in S^\star_m$.

2.2.3. Igazítások

Az **igazítás**okat (*alignment*) legelőször az [55] publikációban vezették be, majd a [105] publikációban továbbfejlesztették. Az [5] publikációban kimutatták, hogy az igazítás kiszámításának sebessége nagyban függ az alapul szolgáló keresési algoritmus megfelelő paraméterezésétől, és a [106] publikációban ajánlásokat tettek a paraméterek konfigurációjára is a számítási hatékonyság növelése érdekében. A [107] publikációban könnyű és hatékony módszereket mutattak be az offline igazítás számításhoz. Az igazítás számításának online megfelelőjét, az **előtag-igazítás** (*prefix-alignment*) számítását, először a [6] publikációban mutatták be.

Az (előtag-)igazításokat arra használják, hogy a nyomvonalakat egy referencia-folyamatmodellhez viszonyítva magyarázzák, tehát minden egyes nyomvonalat a folyamatmodell egy tüzelési szekvenciájához illesztenek. Az igazítás első sora egy nyomvonalat (azaz eseménynapló lépések sorozatát), a második sor pedig egy folyamatmodell tüzelési szekvenciáját (azaz modell lépések sorozatát) ábrázolja.

Egy igazítás **igazítási mozgás**ok véges sorozatából áll. Ha csak a vezérlésfolyam perspektívát vesszük figyelembe, akkor egy igazításban háromféle mozgás lehet: eseménynapló mozgás, modell mozgás, és szinkron mozgás. Ha több perspektívát is figyelembe veszünk, akkor kétféle szinkronmozgás lehet: helyes és helytelen szinkron mozgás. A nem szinkron mozgások esetében a $\gg$ szimbólum jelöli, hogy az adott igazítási mozgásban nem történt eseménynapló lépés (modell mozgás esetén) vagy modell lépés (eseménynapló mozgás esetén).

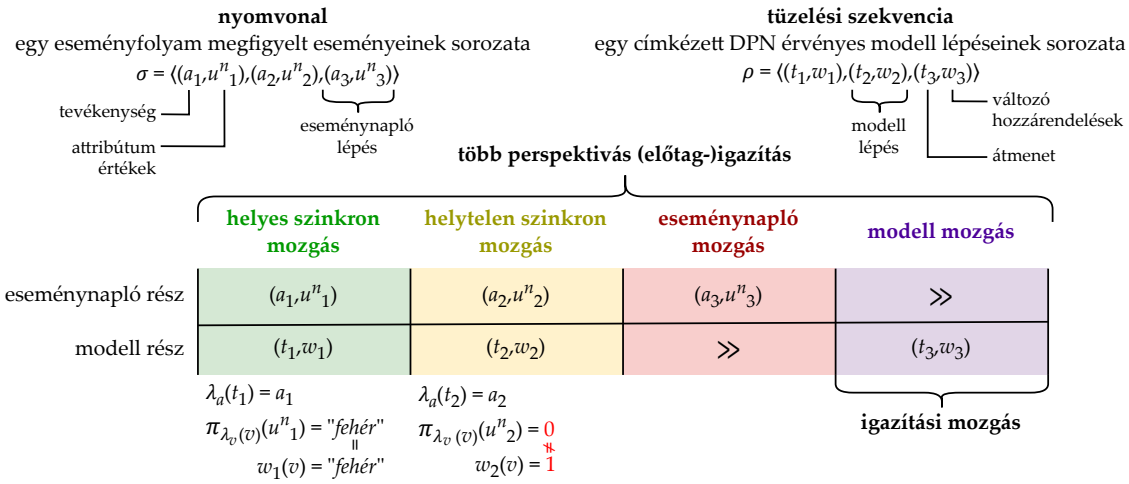
11. Definíció (Vezérlésfolyam igazítási mozgás; Vezérlésfolyam igazítás)**.** Legyen $\mathcal{A}$ a megfigyelhető tevékenységek univerzuma és T egy Petri-háló folyamatmodell átmeneteinek véges halmaza. Legyen λ_a a tevékenységcímke függvény, ami minden $t \in T$ átmenethez visszaadja a megfelelő $a \in \mathcal{A}$ tevékenységet (amennyiben az megfigyelhető) vagy τ-t. Az összes megengedett vezérlésfolyam igazítási mozgást a $\Gamma_c = (\mathcal{A} \cup \{\gg\}) \times (T \cup \{\gg\}) \setminus \{(\gg, \gg)\}$ halmaz adja meg, ahol
- $(a, \gg)$ egy eseménynapló mozgás, ha $a \in \mathcal{A}$,
- $(\gg, t)$ egy modell mozgás, ha $t \in T$,
- (a, t) egy szinkron mozgás, ha $a \in \mathcal{A} \wedge t \in T \wedge \lambda_a(t) = a$,
- $(\gg, \gg)$ egy nem megengedett mozgás.

A $\gamma_c \in \Gamma_c^\star$ egy vezérlésfolyam igazítás, ami megengedett vezérlésfolyam igazítási mozgások véges sorozata.

12. Definíció (Több perspektívás igazítási mozgás; Több perspektívás igazítás)**.** Legyen $\mathcal{A}$ a megfigyelhető tevékenységek univerzuma, S_l a lehetséges eseménynapló lépések halmaza, és S_m az érvényes modell lépések halmaza. Legyen λ_a a tevékenységcímke függvény, ami minden $s_m = (t, w) \in S_m$ érvényes modell lépés $t \in T$ átmenetéhez visszaadja a megfelelő $a \in \mathcal{A}$ tevékenységét (amennyiben az megfigyelhető) vagy τ-t. Az összes megengedett több perspektívás igazítási mozgást a $\Gamma = (S_l \cup \{\gg\}) \times (S_m \cup \{\gg\}) \setminus \{(\gg, \gg)\}$ halmaz adja meg, ahol
- $(s_l, \gg)$ egy eseménynapló mozgás, ha $s_l \in S_l$,
- $(\gg, s_m)$ egy modell mozgás, ha $s_m \in S_m$,
- (s_l, s_m) egy helyes szinkron mozgás, ha $s_l = (a, u^n) \in S_l \wedge s_m = (t, w) \in S_m \wedge \lambda_a(t) = a \wedge \forall_{v \in wr(t)}(\pi_{\lambda_v(v)}(u^n) = w(v))$,
- (s_l, s_m) egy helytelen szinkron mozgás, ha $s_l = (a, u^n) \in S_l \wedge s_m = (t, w) \in S_m \wedge \lambda_a(t) = a \wedge \exists_{v \in wr(t)}(\pi_{\lambda_v(v)}(u^n) \neq w(v))$,
- $(\gg, \gg)$ egy nem megengedett mozgás.

A $\gamma \in \Gamma^\star$ egy több perspektívás igazítás, ami több perspektívás igazítási mozgások véges sorozata.

A lehetséges több perspektívás (előtag-)igazítási mozgásokat a 2.1. ábra szemlélteti. A **szinkronmozgás** egy elvárt viselkedést jelez; egy elvárt tevékenységet végrehajtottak és rögzítettek. A **helyes szinkronmozgás** olyan szinkronmozgás, ahol az eseményattribútumok elvárt értékei rögzítésre kerültek az eseménynaplóban. A **helytelen szinkronmozgás** olyan szinkronmozgás, ahol legalább egy váratlan értéket rögzítettek. Az **eseménynapló mozgás** nem várt viselkedést jelez; olyan tevékenységet, amit nem lett volna szabad végrehajtani, de mégis végrehajtották (vagy legalábbis rögzítésre került). Ez egy modell lépés nélküli eseménynapló lépésként jelenik meg az igazításban. A **modell mozgás** hiányzó viselkedést jelez; olyan tevékenységet, amelyet végre kellett volna hajtani, de nem hajtottak végre (vagy legalábbis nem került rögzítésre). Ez egy eseménynapló lépés nélküli modell lépésként jelenik meg az igazításban.



2.1. ábra. Egy több perspektívás (előtag-)igazítás felépítése

Átalakító függvény

Egy több perspektívás igazítási mozgás átalakítható vezérlésfolyam igazítási mozgássá (információvesztéssel) az attribútumértékek elhagyásával az eseménynapló lépésből és a változó hozzárendelések elhagyásával a modell lépésből. Ennek megfelelően egy több perspektívás igazítás átalakítható egy vezérlésfolyam igazítássá, ha az átalakító függvényt minden igazítási mozgásra alkalmazzuk.

13. Definíció (Több perspektívás igazítást vezérlésfolyam igazítássá átalakító függvény)**.** Az $f_c : \Gamma \to \Gamma_c$ egy több perspektívás igazítási mozgást vezérlésfolyam igazítási mozgássá alakító függvény, ami az átalakítás során

- minden $s_l = (a, u^n) \in S_l$ eseménynapló mozgást a benne levő $a \in \mathcal{A}$ tevékenységre cseréli, valamint
- minden $s_m = (t, w) \in S_m$ modell mozgást a benne levő $t \in T$ átmenetre cseréli.

Az $f_c^\star : \Gamma^\star \to \Gamma_c^\star$ egy γ több perspektívás igazítást γ_c vezérlésfolyam igazítássá alakító függvény, ami az f_c függvényt alkalmazza a γ igazításban lévő összes mozgásra.

Költségfüggvények

Az igazítás-alapú megfelelőség-ellenőrzés célja, hogy olyan optimális igazítást találjon, amely minimalizálja a nyomvonal és a tüzelési szekvencia közötti eltéréseket. Az eltéréseket költségfüggvények alapján pontozzuk, amely alapértelmezett vagy felhasználó által meghatározott lehet. Az alapértelmezett költségfüggvények minden eltéréshez azonos 1-es költséget rendelnek. Ebben a dolgozatban kétfajta költségfüggvényt használunk: **tevékenységköltség-függvény** (*activity cost function*) és **változóköltség-függvény** (*variable cost function*). A tevékenységköltség-függvény egy tevékenység, míg a változóköltség-függvény egy folyamatváltozóhoz kapcsolódó attribútumérték költségét határozza meg.

14. Definíció (Tevékenységköltség-függvény)**.** A $\kappa_a : \Gamma_c \to \mathbb{R}_{\geq 0}$ tevékenységköltség-függvény, minden egyes $(a, t) \in \Gamma_c$ megengedett vezérlésfolyam igazítási mozgáshoz egy nem negatív költséget rendel. Az alapértelmezett tevékenységköltség-függvény csak a szinkron mozgáshoz, valamint egy nem megfigyelhető tevékenységet tartalmazó modell mozgáshoz rendel 0 költség értéket. Minden egyéb igazítási mozgáshoz 1 értéket rendel. Tehát

$$\kappa_a(a,t) = \begin{cases} 0 & \text{ha } (a \in \mathcal{A} \wedge t \in T \wedge \lambda_a(t) = a) \vee (a = \gg \wedge t \in T \wedge \lambda_a(t) = \tau) \\ 1 & \text{különben} \end{cases} \tag{2.2}$$

15. Definíció (Változóköltség-függvény)**.** Legyen $((a, u^n), (t, w)) \in \Gamma$ egy megengedett több perspektívás igazítási mozgás. A $\kappa_v : V \times \mathcal{U} \times \mathcal{U} \to \mathbb{R}_{\geq 0}$ változóköltség-függvény minden egyes $v \in V$ folyamatváltozó elnevezés, $u_i \in \mathcal{U}_i$ eseménynapló szerinti érték és $w(v) \in val(v)$ modell szerinti érték rendezett hármashoz (feltéve, hogy $i = \lambda_v(v)$ és $\pi_i(u^n) = u_i$) egy nem negatív költséget rendel. Az alapértelmezett változóköltség-függvény csak azokhoz a rendezett hármasokhoz rendel 0 értéket, ahol az eseménynapló szerinti érték és modell szerinti érték megegyezik (azaz $u_i = w(v)$), minden más rendezett hármashoz 1 értéket rendel. Tehát

$$\kappa_v(v, u_i, w(v)) = \begin{cases} 0 & \text{ha } v \in V \wedge i = \lambda_v(v) \wedge u_i \in \mathcal{U}_i \wedge w(v) \in val(v) \wedge u_i = w(v) \\ 1 & \text{különben} \end{cases} \tag{2.3}$$

Ha az igazítási mozgás modell mozgás (azaz nem tartalmaz eseménynapló lépést) akkor $u_i = \varnothing$ és a „különben" eset teljesül.

Egy **vezérlésfolyam igazítás költsége** az azt tartalmazó összes igazítási mozgás tevékenységköltség-függvény által meghatározott költségének az összege:

$$K_c(\gamma_c) = \sum_{(a,t) \in \gamma_c} \kappa_a(a, t) \tag{2.4}$$

Egy **több perspektívás igazítás költsége** az azt tartalmazó összes igazítási mozgás tevékenységköltség-függvény és változóköltség-függvény által meghatározott költségének az összege. Egy több perspektívás igazítási mozgás tevékenységköltségének a kiszámításához az igazítási mozgást vezérlésfolyam igazítási mozgássá alakítjuk. Egy igazítási mozgás változóköltsége az átmenet által írt minden változó változóköltségének az összege, ami csak modell lépést tartalmazó igazítások esetén kerül kiszámításra. Tehát egy több perspektívás igazítás költségét a következő képlettel lehet kiszámolni:

$$K(\gamma) = K_c(f_c^\star(\gamma)) + \sum_{((a,u^n),(t,w)) \in \gamma} \sum_{v \in dom(w)} \kappa_v(v, u_i, w(v)) \tag{2.5}$$

2.2.4. Optimális változó hozzárendelési probléma és annak gyorsítótárazása

Az **optimális változó hozzárendelési probléma** (*Optimal Variable Assignment Problem*, röviden *OVAP*) célja megtalálni az **optimális változó hozzárendelés**t (*Optimal Variable Assignment*, röviden *OVA*), azaz azon érvényes értékeket a folyamatváltozó írási műveletekhez, amikkel minimalizálható a több perspektívás igazítás teljes változóköltsége.

Az OVAP átalakítható olyan vegyes egészértékű lineáris programozási (*Mixed-Integer Linear Programming*, röviden *MILP*) problémává [108, 109], ahol a korlátok az igazításban szereplő átmenetek változóírási műveleteiből és őrkifejezéseiből épülnek fel (5. algoritmus). A teljes változóköltség úgy minimalizálható, ha a megfigyelt események attribútum értékei és az adat Petri-háló folyamatmodell által megkövetelt változó hozzárendelések közötti eltérések költségét minimalizáljuk. Ezért a célfüggvény a helytelen és a hiányzó folyamatváltozó hozzárendelések összege. Az OVAP részletesebb leírása az [57] publikációban olvasható.

OVAP gyorsítótár

Az OVAP megoldása az optimális több perspektívás (előtag-)igazítás előállításának a legszámításigényesebb része, ezért az OVAP megoldások tárolására létrehoztam egy **OVAP gyorsítótár**at.

Az OVAP előállításához egy úgynevezett **változó írási szekvencia** kerül létrehozásra (4. algoritmus). A változó írási szekvencia a tüzelési szekvenciából csak azokat a modell lépéseket tartalmazza, amikben olyan átmenet van, amihez őrkifejezés vagy változó írási műveletek tartoznak. Azok a modell lépések, amikhez egyik sem tartozik, az OVAP szempontjából nem számítanak. A folyamatváltozókba írt értékek a nyomvonal alapján kerülnek megállapításra. Fontos, hogy csak az érvényes (a változó értékhalmazába tartozó) értéket rendelhetünk a változóhoz. Ha a változóhoz nem lett megfigyelve érték vagy a megfigyelt érték érvénytelen, akkor üres ($\varnothing$) értéket rendelünk hozzá.

A változó írási szekvencia egyértelműen azonosítja az OVAP-ot, így az használható az OVAP helyett az OVAP megoldásának gyorsítótárazásához. Az **OVAP megoldása** változó hozzárendelések véges sorozataként van kifejezve. A gyorsítótárban az OVAP megoldása mellett az előállítandó több perspektívás előtag-igazításhoz tartozó **változóköltség** és a **folyamatváltozók aktuális értéke** is eltárolásra kerül, mivel azoknak az értéke csak változó írásoktól függ (a vezérlésfolyamtól nem).

16. Definíció (Változó írási szekvencia; OVAP megoldása; OVAP gyorsítótár)**.** Legyen T egy adat Petri-háló átmeneteinek véges halmaza és W az adat Petri-háló lehetséges érvényes változó hozzárendeléseinek halmaza. A $\overline{\rho} = \langle (t_1, w_1), (t_2, w_2), \ldots \rangle$ változó írási szekvencia modell lépések véges sorozata, azaz $\overline{\rho} \in (T \times W)^\star$. Az $\omega = \langle w_1, w_2, \ldots \rangle$ OVAP megoldása változó hozzárendelések véges sorozata, azaz $\omega \in W^\star$. Az Ω OVAP gyorsítótárat az $\Omega : (T \times W)^\star \to W^\star \times \mathbb{R}_{\geq 0} \times W$ függvénnyel definiáljuk. Ha ω a $\overline{\rho}$ által aznosított OVAP megoldása, akkor a hosszuk megegyezik (azaz $|\overline{\rho}| = |\omega|$), valamint a változó hozzárendelésekben érintett változók megegyeznek (minden $i \in 1, 2, \ldots, |\overline{\rho}|$ esetén $dom(\pi_2(\pi_i(\overline{\rho}))) = dom(\pi_i(\omega))$). Továbbá, $\Omega(\overline{\rho}) = (\omega, g_v, \alpha)$, ahol g_v a változóköltség és α a folyamatváltozók aktuális értéke ω használata esetén.

2.2.5. Szinkron szorzat háló

Az optimális igazítás megtalálásának problémája a legrövidebb út problémájára redukálható. Az optimális (előtag-)igazítás kiszámításának egyik lehetséges módja a szinkron szorzat háló (*Synchronous Product Net*, röviden *SPN*) állapotterében a legrövidebb út megtalálása [105].

Az SPN egy nyomvonalháló (*trace net*, egy nyomvonal vezérlésfolyam részéből felépített Petri-háló) és egy munkafolyamat-háló (*workflow net*) kompozíciója, amelyben minden egyes átmenet egy igazítási mozgásnak felel meg. Ennélfogva, az átmenetek bármely sorozata megfelel egy (előtag-)igazításnak. A keresést a költségek irányítják, amelyeket az egyes átmenetekhez a megadott tevékenységköltség-függvény szerint rendelünk. Az optimális (előtag-)igazítás megtalálásához a legrövidebb utat (azaz a minimális összköltségű igazítási mozgások sorozatát) keressük az SPN kezdeti jelölésétől egy olyan jelölésig, ahol az SPN nyomvonalhálós részének utolsó (legutóbb hozzáadott) helyén jelölés van.

Az SPN állapottér az optimális több perspektívás (előtag-)igazítás megtalálására is felhasználható. A lehetséges változó hozzárendelések kombinációja azonban akár végtelen is lehet, ezért a keresési tér minimálisan tartása érdekében a DPN modell vezérlésfolyam másolatát (azaz a Petri-háló modellt) használjuk az SPN felépítéséhez. Az OVA-k egy igazítási mozgáshoz a kiválasztott előd- és utódlépéstől is függhetnek, így azokat újra kell számítani, amikor az útvonal megváltozik.

17. Definíció (Szinkron szorzat háló)**.** Legyen σ egy nyomvonal, $N^\sigma = (P^\sigma, T^\sigma, F^\sigma, [p_i^\sigma], [p_f^\sigma], \lambda_a^\sigma)$ a hozzá tartozó nyomvonalháló és $N = (P, T, F, [p_i], [p_f], \lambda_a)$ egy munkafolyamat-háló, ahol $P^\sigma \cap P = \emptyset$ és $F^\sigma \cap F = \emptyset$. A $N^S = (P^S, T^S, F^S, M_i^S, M_f^S, \lambda_a^S)$ egy SPN, ahol

- $P^S = P^\sigma \cup P$,
- $T^S = (T^\sigma \times \{\gg\}) \cup (\{\gg\} \times T) \cup \{(t^\sigma, t) \in T^\sigma \times T \mid \lambda_a(t) = \lambda_a^\sigma(t^\sigma) \neq \tau\}$,
- $F^S = \{(p, (t^\sigma, t)) \in P^S \times T^S \mid (p, t^\sigma) \in F^\sigma \vee (p, t) \in F\} \cup \{((t^\sigma, t), p) \in T^S \times P^S \mid (t^\sigma, p) \in F^\sigma \vee (t, p) \in F\}$,
- $M_i^S = [p_i^\sigma, p_i]$,
- $M_f^S = [p_f^\sigma, p_f]$,
- $\lambda_a^S : T^S \to (\mathcal{A} \cup \{\tau, \gg\}) \times (\mathcal{A} \cup \{\tau, \gg\})$ (feltételezve, hogy $\gg \notin \mathcal{A} \cup \{\gg\}$), ahol
  - $\lambda_a^S(t^\sigma, \gg) = (\lambda_a^\sigma(t^\sigma), \gg)$, ha $t^\sigma \in T^\sigma$,
  - $\lambda_a^S(\gg, t) = (\gg, \lambda_a(t))$, ha $t \in T$,
  - $\lambda_a^S(t^\sigma, t) = \lambda_a^S(\lambda_a^\sigma(t^\sigma), \lambda_a(t))$, ahol $t^\sigma \in T^\sigma, t \in T$.

18. Definíció (Szinkron szorzat háló állapottere, állapota)**.** Legyen $N^S = (P^S, T^S, F^S, M_i^S, M_f^S, \lambda_a^S)$ egy szinkron szorzat háló (SPN). Az N^S SPN összes elérhető jelölését (más néven állapotterét) egy M_i kezdeti állapot jelölés esetén az $\mathcal{R}(N^S, M_i^S)$ definiálja. Az N^S SPN egy lehetséges jelölését (állapotát) az $m \in \mathcal{R}(N^S, M_i^S)$ jelöli.

2.2.6. Inkrementális A* algoritmus

A [7] publikációban bemutatott inkrementális A* algoritmus a jól ismert A* útvonalkeresési algoritmus ([110]) módosított változata. Az A* algoritmus leginkább kis folyamatmodelleken történő igazítások kiszámítására alkalmas, nagyobb modellek esetén általában a szimbolikus technika teljesít jobban [111]. A szimbolikus technika azonban csak vezérlésfolyam igazítás számítására használható, mivel az előretekintő keresés mellett visszafelé történő keresést is végez és az optimális több perspektívás

(előtag-)igazításokat csak előretekintő keresés használatával lehet megtalálni (mivel a változók értéke befolyásolhatja az elérhető átmeneteket). További probléma, hogy csak az alapértelmezett tevékenységköltség-függvény használatát engedi.

Az inkrementális A* algoritmus egy informált keresési algoritmus, amely egy rögzített kezdeti és egy inkrementálisan változó végállapothoz keresi a legrövidebb utat. Ez használható egy eseményfolyamból származó befejezetlen nyomvonal optimális előtag-igazításának a megtalálására. A keresési tér a korábban tárgyalt SPN állapottér (2.2.5. alszakasz), ami kezdetben csak a modell mozgásokat tartalmazza. Az új események megfigyelésekor ez a keresési tér eseménynapló mozgásokkal és szinkronmozgásokkal bővül.

Az inkrementális A* algoritmus lépései röviden a következők:

1. Egy új esemény (esetazonosító és tevékenység rendezett pár) megfigyelése.
2. Az SPN-t az adott esetazonosítóhoz tartozó folyamatpéldányra kiterjesztjük az új tevékenységgel. Amikor egy SPN-t új tevékenységgel bővítünk, új átmenetek kerülnek hozzáadásra, amelyek az új tevékenységen történő eseménynapló mozgást és a lehetséges szinkronmozgást képviselik.
3. A legrövidebb út keresése a kibővített SPN állapottérben folytatódik a korábban gyorsítótárazott keresési eredményekből, azaz a már feltárt/megvizsgált állapotokból kiindulva.
4. Az algoritmus az új előtag-igazítást visszaadja és az újabb keresési eredmények gyorsítótárba kerülnek.

Az algoritmus esetazonosítónként gyorsítótárazza a keresést megkönnyítő objektumokat, amelyek a következők:

- σ: nyomvonal (tevékenységek sorozata), $\sigma \in \mathcal{A}^\star$;
- O: nyitott halmaz, a még megvizsgálandó állapotok halmaza, $O \subseteq \mathcal{R}(N^S, M_i^S)$;
- C: zárt halmaz, a már megvizsgált állapotok halmaza, $C \subseteq \mathcal{R}(N^S, M_i^S)$;
- g: eddigi költség függvény, egy $m \in \mathcal{R}(N^S, M_i^S)$ SPN állapothoz az addig ismert teljes költséget visszaadó függvény, $g : \mathcal{R}(N^S, M_i^S) \to \mathbb{R}_{\geq 0}$;
- μ (a [7] publikációban p): előzmény függvény, ami egy $m' \in \mathcal{R}(N^S, M_i^S)$ állapotba vezető $(t^\sigma, t) \in T^S$ átmenet és $m \in \mathcal{R}(N^S, M_i^S)$ állapot rendezett párját visszaadó függvény, $\mu : \mathcal{R}(N^S, M_i^S) \to T^S \times \mathcal{R}(N^S, M_i^S)$;
- $\overline{\gamma}$: igazítás függvény, egy $m \in \mathcal{R}(N^S, M_i^S)$ állapothoz az aktuális előtag-igazítást visszaadó függvény, $\overline{\gamma} : \mathcal{R}(N^S, M_i^S) \to \Gamma_c^\star$.

Az algoritmus a keresés folyamán a nyitott halmazból mindig a legkisebb teljes becsült költséggel (f értékkel) rendelkező állapotot választja ki vizsgálatra. Az f értékét egy m állapotra a következőképpen számítja ki:

$$f(m) = g(m) + h(m) \tag{2.6}$$

ahol m az SPN egy állapota ($m \in \mathcal{R}(N^S, M_i^S)$), g az addig ismert igazítási költséget visszaadó függvény ($g : \mathcal{R}(N^S, M_i^S) \to \mathbb{R}_{\geq 0}$) és h egy heurisztikus függvény, ami a becsült fennmaradó költséget adja vissza ($h : \mathcal{R}(N^S, M_i^S) \to \mathbb{R}_{\geq 0}$).

Mivel az átmenetek vezérlésfolyam (előtag-)igazítás mozgásoknak felelnek meg, a g értékét a κ_a (a [7] publikációban d) tevékenységköltség-függvény használatával számítja ki.

Az (előtag-)igazítás számításához használt heurisztika a hagyományos igazítás számításához használt heurisztika [105] relaxált verziója. A heurisztika megfogalmazható egy egészértékű lineáris programozási (*Integer Linear Programming*, röviden *ILP*) problémaként, ahol minden SPN átmenethez egy változó tartozik, amely az

adott átmenet tüzeléseinek számát írja le. A célunk az aktuális végállapotba vezető legrövidebb (legkisebb teljes költséggel rendelkező) útvonal megtalálása az SPN-ben, ezért a célfüggvény a változók tevékenységköltséggel megszorzott értékeinek összegét minimalizálja. A h értéke becsült érték, mivel a végállapot még változhat (a nyomvonalháló változása esetén) és az algoritmus csak a tevékenységköltségekkel számol, a változóköltségekkel nem.

Módosítások

Az inkrementális A* algoritmust módosítottam, hogy optimális több perspektívás előtag-igazítás megtalálására is alkalmas legyen. A módosított verzióról részletes leírás a 2.3. szakaszban olvasható. Itt csak a különbségek kerülnek röviden kiemelésre.

A módosított algoritmus az eddig említetteken felül az α változó értékek függvényt gyorsítótárazza, ami egy $m \in \mathcal{R}(N^S, M_i^S)$ állapothoz a folyamatváltozók aktuális értékét visszaadó függvény ($\alpha : \mathcal{R}(N^S, M_i^S) \to W$). További különbségek a gyorsítótárakban:

- σ: kiterjesztett (nem csak a tevékenységeket tartalmazó) nyomvonal, $\sigma \in S_l^\star$;
- $\overline{\gamma}$: igazítás függvény, egy $m \in \mathcal{R}(N^S, M_i^S)$ állapothoz az aktuális több perspektívás előtag-igazítást visszaadó függvény, $\overline{\gamma} : \mathcal{R}(N^S, M_i^S) \to \Gamma^\star$.

Az f érték számításához ugyanaz a képlet van használva. Azonban, mivel az átmenetek több perspektívás (előtag-)igazítás mozgásoknak felelnek meg, a g értékét a κ_a tevékenységköltség-függvény mellett a κ_v változóköltség-függvény is meghatározza.

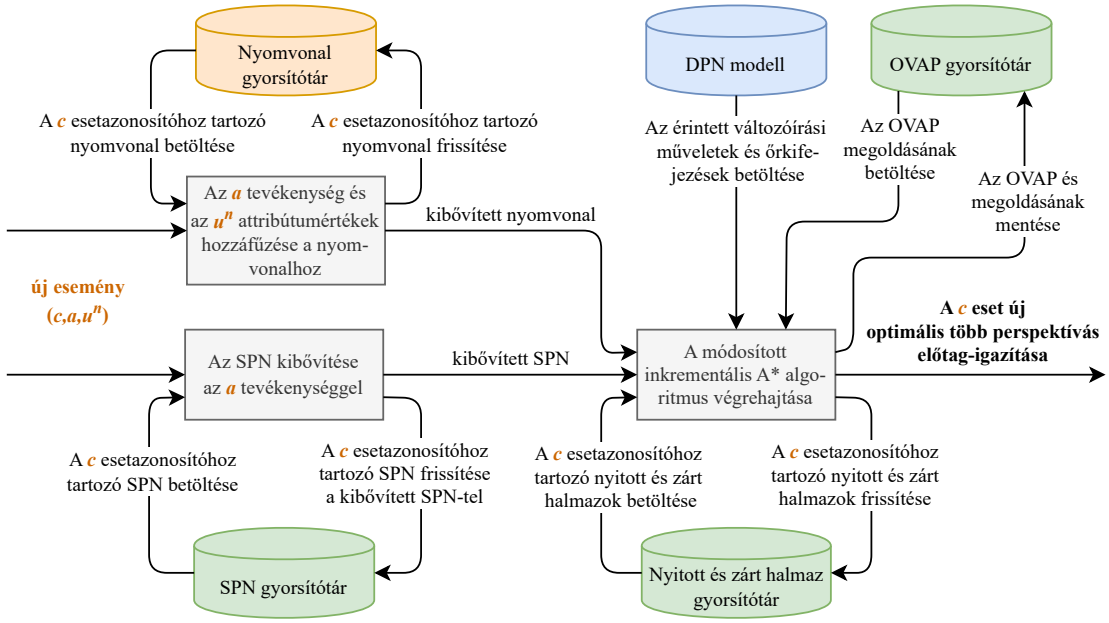
2.3. A javasolt MOCC módszer

Ebben a szakaszban a javasolt MOCC módszer kerül bemutatásra. Először a javasolt megközelítés egészének áttekintése, majd az összetevők részletesebb leírása következik.

2.3.1. Áttekintés

A javasolt megoldás lényegében a kiegyensúlyozott több perspektívás megfelelőség-ellenőrzési módszer [13] és a [7] publikációban bemutatott online megfelelőség-ellenőrzési algoritmus ötvözetének tekinthető. Azért az említett online megfelelőség-ellenőrzési módszert választottam, mert jelenleg az a legpontosabb és egyben a leggyorsabb megoldás, amely több perspektívára is kiterjeszthető. A javasolt megoldás egy DPN-folyamatmodell, egy eseményfolyam és a felhasználó által meghatározott tevékenység- és változóköltség-függvények birtokában minden alkalommal, amikor egy esethez új esemény kerül megfigyelésre, kiszámítja az optimális több perspektívás előtag-igazítást. A [7] publikációban bemutatott algoritmushoz hasonlóan az algoritmus lényege, hogy felhasználja az SPN állapottér feltárásának korábban kiszámított eredményeit. Minden egyes folyamatpéldányhoz (esethez) egy nyomvonal és egy SPN kerül eltárolásra. A nyomvonal az eddig megfigyelt eseményekből áll, az SPN pedig a folyamatmodell és a nyomvonal vezérlésfolyam másolatából épül fel. A teljes nyomvonalat el kell tárolni, mivel az SPN csak a tevékenységneveket tartalmazza.

A javasolt MOCC módszer áttekintése a 2.2. ábrán látható és a működését leíró fő algoritmus a 2.3.2. alszakaszban olvasható. Ha egy új esemény kerül megfigyelésre, a nyomvonal kibővül ezzel az új eseménnyel, és az SPN kibővül a tevékenységgel (azaz az eseményben szereplő tevékenységnévvel). Ezután a keresés a módosított inkrementális A* algoritmussal folytatódik az előző keresésből származó, előre kitöltött nyitott és zárt halmazok felhasználásával (2.3.3. alszakasz). A nyitott halmaz minden egyes állapotához megvizsgálja a lehetséges utódállapotokat: a vezérlésfolyam igazítást rekonstruálja és kibővíti a lehetséges következő igazítási mozgásokkal, majd OVA-kat számol, hogy a kibővített vezérlésfolyam igazításokat kiegészítse velük (2.3.4. alszakasz). Az OVA-k kiszámításához szükség van az igazításban, pontosabban a tüzelési szekvenciában lévő átmenetek változóírási műveleteire és őrkifejezéseire is. Az utódállapotok közül az OVA-kkal és minimális összköltséggel (azaz minimális tevékenység és változóköltséggel) rendelkező állapotot választja ki. Ezáltal optimális több perspektívás (előtag-)igazítást ad vissza.



2.2. ábra. A javasolt MOCC módszer áttekintése

A számítási idő csökkentése érdekében az algoritmust két funkcióval bővítettem:

- **Közvetlen szinkronizálás**: A [80] publikációban bemutatott közvetlen szinkronizáláshoz hasonlóan az ötlet lényege, hogy kihagyjuk a legrövidebb útvonal keresését azokban az esetekben, amikor a korábban kiszámított előtag-igazítás egyszerűen kibővíthető egy olyan szinkronmozgással, amely magában foglalja az újonnan megfigyelt eseményt. A több perspektívás esetben azonban a legrövidebb út keresése csak akkor hagyható ki, ha a szinkronmozgás helyes (azaz minden változó írása helyes és az érintett átmenet őre kielégíthető). Ha a közvetlen szinkronizálás lehetséges, akkor a legrövidebb út megtalálására irányuló heurisztikus újraszámítások (amelyek nagy számítási igénnyel járhatnak) elkerülhetők, és így felgyorsítható az előtag-igazítás számítása.

- **OVAP megoldások gyorsítótárazása**: Az OVA-k kiszámítása az egész algoritmus legidőigényesebb része, ezért a módszer online környezetben való al-

kalmazhatósága szempontjából kritikus fontosságú az erre fordított idő csökkentése. Feltételezve, hogy a lehetséges változó hozzárendelési problémák halmaza véges (azaz nem minden esetnek van egyedi nyomvonala), a probléma megoldása a probléma- és megoldáspárok gyorsítótárazásával felgyorsítható. Ha egy probléma ismételten előfordul, akkor az eredményt a gyorsítótárból lehet előhívni, és az OVAP megoldásához szükséges számítások kihagyhatók.

A javasolt MOCC módszer két legnagyobb számítási igényű része tehát a módosított inkrementális A* algoritmus alkalmazása és az OVAP megoldása. A módosított inkrementális A* algoritmus legrosszabb esetben exponenciálisan bonyolult az útvonal hossza tekintetében, amely a kiinduló keresési pontból a legközelebbi célpontba vezet. Az OVAP megoldásának legrosszabb esetben tapasztalható komplexitása pedig exponenciálisan növekszik az érintett változók és az átmenetekhez kapcsolódó őrök számával. Egy jól optimalizált OVAP gyorsítótárral azonban az OVA számítási komplexitása a legrosszabb esetben is lineárisan függ a gyorsítótár méretétől. Ha közvetlen szinkronizálás lehetséges, a módosított inkrementális A* algoritmus alkalmazása és az OVA számítás teljes mértékben elhagyható.

Az esetek gyorsítótárazása

A fent bemutatott MOCC módszer csak online igazításokat számol, ami bizonyos folyamatok esetében nem ad reális képet a folyamat lefutásáról. Ha egy adott idő után egy esethez nem kerül új esemény megfigyelésre, akkor azt befejezettnek kell tekinteni, és offline igazítást kell hozzá kiszámítani. (Az offline itt azt jelenti, hogy az igazítás kiszámítása során a folyamat lefutást befejezettnek tekintjük.) Ha például egy esetnek hiányos a nyomvonalának a vége, akkor csak az offline igazítás képes azon eseményeket kimutatni, amik hiányoznak a helyes befejezéshez. A javasolt MOCC módszer másik problémája, hogy az eseteket nem hatékonyan gyorsítótárazza. A gyorsítótárakban minden esetet megtart, amihez valaha legalább egy esemény megfigyelésre került, így sok folyamatpéldány esetén drasztikusan megnőhet a lefoglalt memória mérete.

Az említett két probléma megoldására az MOCC módszer keretében létrehoztam egy speciális eset gyorsítótárazási módszert. Ehhez két további gyorsítótárat vezettem be: egyet a folyamatban lévő esetek számára (rugalmas és korlátlan mérettel), és egy másikat a befejezett esetek számára (szintén rugalmas, de korlátozott mérettel). A folyamatban lévő esetek gyorsítótárához hozzáadott esetekhez online igazítás, míg a befejezett esetek gyorsítótárába kerülő esetekhez offline igazítás kerül kiszámításra. A módszer lényege, hogy az eseteket alapból a folyamatban lévő esetek gyorsítótárában tároljuk. Ha egy eset adott időtartamon belül nem frissül, akkor azt a befejezett esetek gyorsítótárába „félrerakjuk”. Ha egy befejezettnek vélt esethez újabb esemény kerül megfigyelésre, akkor az visszakerül a folyamatban lévő esetek gyorsítótárába. Ha egy „félrerakott” esethez hosszabb ideig nem kerül újabb esemény megfigyelésre, akkor idővel (a gyorsítótár méretétől függően) „kiszorul” az újabban „félrerakott” esetek által. A módszerről bővebben a 2.3.5. alszakaszban írok.

2.3.2. A fő algoritmus

Ebben az alszakaszban a fő algoritmus (1. algoritmus) kerül bemutatásra. Az algoritmus bemenetként egy N_λ címkézett DPN folyamatmodellt, egy E eseményfolyamot, egy felhasználó által definiált κ_a tevékenységköltség-függvényt, egy felhasználó által definiált κ_v változóköltség-függvényt, valamint egy Ω OVAP gyorsítótárat vár. A költségfüggvények megadása opcionális; ha nem adjuk meg őket, akkor az alapértelmezett költségfüggvényeket használja (2–5. sor). Az Ω megadása szintén opcionális; ha nem adjuk meg, akkor egy új, kezdetben üres gyorsítótárat fog használni (6–7. sor).

Algoritmus 1: A MOCC fő algoritmus (Inkrementális több perspektívás előtag-igazítás számítása)

```
   input : N_λ = (P, T, F, V, val, in, wr, gd, [p_i], [p_f], λ_a, λ_v), E ∈ E^⋆, κ_a : Γ_c → ℝ_≥0,
           κ_v : V × U × U → ℝ_≥0, Ω : (T × W)^⋆ → W^⋆ × ℝ_≥0 × W
1  begin
2      if κ_a nincs definiálva then
3          legyen κ_a : Γ_c → ℝ_≥0 ;                  // alapértelmezett tevékenységköltség-függvény létrehozása
4      if κ_v nincs definiálva then
5          legyen κ_v : V × U × U → ℝ_≥0 ;             // alapértelmezett változóköltség-függvény létrehozása
6      if Ω nincs definiálva then
7          legyen Ω : (T × W)^⋆ → W^⋆ × ℝ_≥0 × W ;                  // OVAP gyorsítótár létrehozása
8      i ← 1 ;
9      while true do
10         e ← E(i) ;                                  // az eseményfolyam i-edik eseményének kinyerése
11         c ← π_1(e) ;                                // az esetazonosító kinyerése az aktuális eseményből
12         a ← π_2(e) ;                                // a tevékenység kinyerése az aktuális eseményből
13         u^n ← π_3(e) ;                              // az attribútum értékek kinyerése az aktuális eseményből
14         if c ∉ dom(D_σ) ∧ c ∈ C then                // gyorsítótárak inicializálása az új c esethez
15             D_σ(c) ← ⟨⟩ ;                           // nyomvonal inicializálása
16             D_γ(c) ← ⟨⟩ ;                           // igazítás inicializálása
17             D_C(c) ← ∅ ;                            // zárt halmaz inicializálása
18         D_σ(c) ← D_σ(c) · ⟨(a, u^n)⟩ ;              // az eset nyomvonalának kibővítése
19         legyen N^S = (P^S, T^S, F^S, M_i^S, M_f^S, λ_a^S), N_λ és π_1^⋆(D_σ(c)) alapján;
                                                       // SPN létrehozása/bővítése
20         legyen h : R(N^S, M_i^S)→ℝ_≥0 ;             // heurisztikus függvény definiálása
21         if |D_σ(c)| = 1 then                        // inicializálás az első futtatáshoz a c esetre vonatkozóan
22             D_O(c) ← {M_i^S} ;                      // nyitott halmaz inicializálása
23             D_g(c) ← M_i^S ↦ 0 ;                    // eddigi költség függvény inicializálása
24             D_μ(c) ← M_i^S ↦ (null, null) ;         // előzmény függvény inicializálása
25             D_α(c) ← M_i^S ↦ ⋃_{v∈V} {v ↦ in(v)} ;  // változó értékek függvény inicializálása
26             D_γ̄(c) ← M_i^S ↦ ⟨⟩ ;                   // igazítás függvény inicializálása
27         D_γ(c), D_γ̄(c), D_α(c), D_O(c), D_C(c), D_g(c), D_μ(c), Ω ←
             modA^⋆_inc(N_λ, D_σ(c), a, u^n, N^S, D_O(c), D_C(c), D_g(c), D_μ(c), h, D_α(c), D_γ̄(c), κ_a, κ_v, Ω) ;
                                                       // a legrövidebb útvonal keresésének végrehajtása/folytatása
28         i ← i + 1 ;
```

Az algoritmus az E eseményfolyam minden eseményét a megfigyelés sorrendjében dolgozza fel. Mivel az eseményfolyamot végtelennek feltételezzük, az események feldolgozása egy végtelen ciklusban történik (9. sor). Egy új e esemény megfigyelését követően először a c esetazonosítót, az a tevékenységazonosítót és az u^n attribútumértékeket kivonja az eseményből (10–13. sorok), majd az a-ból és az u^n-ből képzett eseménynapló lépést hozzáfűzi az esethez tartozó nyomvonalhoz (18. sor). Ha ez az e esemény a c eset első megfigyelt eseménye, akkor az a tevékenységből és az N_λ Petri-háló vezérlésfolyam részéből egy új N^S SPN-t készít (19. sor), valamint inicializálja a gyorsítótárakat (15–17. sorok és 22–26. sorok). Ellenkező esetben a korábban konstruált N^S SPN-t bővíti az a tevékenységgel (19. sor). Ezt követően az SPN állapottérhez egy h heurisztikus függvényt definiál (20. sor), majd a módosított

inkrementális $A^\star$ algoritmus meghívásával kiszámítja a c esethez γ-t, az új optimális több perspektívás előtag-igazítást (27. sor). Ezt követően a következő esemény feldolgozása következik.

2.3.3. A módosított inkrementális A* algoritmus

Ebben az alszakaszban a módosított inkrementális $A^\star$ algoritmus (2. algoritmus) kerül bemutatásra, ami az optimális több perspektívás előtag-igazítás kiszámítására használatos.

Algoritmus 2: $modA^\star_{inc}$ (Módosított inkrementális $A^\star$ algoritmus)

input : $N_\lambda = (P, T, F, V, val, in, wr, gd, [p_i], [p_f], \lambda_a, \lambda_v)$, $\sigma \in (\mathcal{A} \times \mathcal{U}^n)^\star$, $a \in \mathcal{A}$, $u^n \in \mathcal{U}^n$,
$N^S = (P^S, T^S, F^S, M_i^S, M_f^S, \lambda_a^S)$, O, $C \subseteq \mathcal{R}(N^S, M_i^S)$, $g : \mathcal{R}(N^S, M_i^S) \to \mathbb{R}_{\geq 0}$,
$\mu : \mathcal{R}(N^S, M_i^S) \to T^S \times \mathcal{R}(N^S, M_i^S)$, $h : \mathcal{R}(N^S, M_i^S) \to \mathbb{R}_{\geq 0}$, $\alpha : \mathcal{R}(N^S, M_i^S) \to W$,
$\overline{\gamma} : \mathcal{R}(N^S, M_i^S) \to \Gamma^\star$, $\kappa_a : \Gamma_c \to \mathbb{R}_{\geq 0}$, $\kappa_v : V \times \mathcal{U} \times \mathcal{U} \to \mathbb{R}_{\geq 0}$, $\Omega : (T \times W)^\star \to W^\star \times \mathbb{R}_{\geq 0} \times W$

1 **begin**
2 legyen $p_{|\sigma|}$ a N^S SPN nyomvonalháló részének utolsó helye;
3 **forall** $m \in \mathcal{R}(N^S, M_i^S) \setminus O \cup C$ **do** // feltáratlan állapotok inicializálása
4 $g(m) \leftarrow \infty$;
5 $f(m) \leftarrow \infty$;
6 **forall** $m \in O$ **do**
7 $f(m) = g(m) + h(m)$; // heurisztika újraszámítása és az f-értékek frissítése
8 **while** $O \neq \emptyset$ **do**
9 $m \leftarrow \arg\min_{m \in O} f(m)$; // minimális f-értékkel rendelkező állapot kivétele O-ból
10 **if** $p_{|\sigma|} \in_+ m$ **then**
11 **return** $\overline{\gamma}(m), \overline{\gamma}, \alpha, O, C, g, \mu, \Omega$;
12 $C \leftarrow C \cup \{m\}$;
13 $O \leftarrow O \setminus \{m\}$;
14 $dsp = false$; // közvetlen szinkronizálás lehetőségének ellenőrzése
15 **if** $\exists (t^\sigma, t) \in T^S \wedge (N^S, m)[(t^\sigma, t)\rangle (N^S, m') \wedge \lambda_a^S((t^\sigma, t)) = (a, a)$ **then**
16 $dsp = true$;
17 $V' \leftarrow \{\}$;
18 **forall** $v \in wr(t)$ **do** // prímváltozók ellenőrzése és beállítása
19 **if** $\pi_{\lambda_v(v)}(u^n) \in val(v)$ **then**
20 $w(v') \leftarrow \pi_{\lambda_v(v)}(u^n)$;
21 $V' \leftarrow V' \cup v'$;
22 **else**
23 $dsp = false$;
24 **if** $dsp = true$ **then** // őrkifejezés ellenőrzése
25 $\alpha' = \alpha(m) \cup w$;
26 $dsp = eval_{V \cup V', val}(gd(t), \alpha')$;
27 **if** $dsp = true$ **then** // közvetlen szinkronizálás végrehajtása, ha lehetséges
28 $O \leftarrow O \cup \{m'\}$; // az új m' állapot hozzáadása
29 $\overline{\gamma}(m') \leftarrow \overline{\gamma}(m) \cdot \langle((a, u^n), (t, w))\rangle$; // a γ bővítése az új mozgással
30 $g(m') \leftarrow g(m)$; // az m' eléréséhez szükséges költség mentése
31 $\mu(m') \leftarrow ((t^\sigma, t), m)$; // az m' előzményének mentése
32 **forall** $v' \in dom(w)$ **do**
33 $\alpha(m')(v) \leftarrow w(v')$; // az érintett változók új értékének mentése
34 **return** $\overline{\gamma}(m'), \overline{\gamma}, \alpha, O, C, g, \mu, \Omega$;
35 **forall** $(t^\sigma, t) \in T^S \wedge (N^S, m)[(t^\sigma, t)\rangle (N^S, m')$ **do**
36 **if** $m' \notin C$ **then** // az m utódállapotainak megvizsgálása
37 $\gamma_c \leftarrow f_c^\star(\overline{\gamma}(m)) \cdot \langle(a, t)\rangle$; // a γ_c bővítése az új mozgással
38 $\gamma', g', \alpha', X \leftarrow getMPPA(N_\lambda, \sigma, \gamma_c, \kappa_a, \kappa_v, \Omega)$; // OVA-s γ megszerzése
39 **if** $\gamma' \neq \langle\rangle$ **then** // a γ exists for γ_c
40 **if** $\exists n \in O \wedge n = m' \wedge g(n) > g'$ **then** // m'-hez egy olcsóbb utat talált
41 $O \leftarrow O \setminus \{n\}$; // a régi n állapot eltávolítása
42 **if** $m' \notin O$ **then**
43 $O \leftarrow O \cup \{m'\}$; // az új m' állapot hozzáadása
44 $\overline{\gamma}(m') \leftarrow \gamma'$; // az új γ mentése
45 $g(m') \leftarrow g'$; // az m' eléréséhez szükséges költség mentése
46 $f(m') \leftarrow g(m') + h(m')$; // az f-érték kiszámolása és mentése
47 $\mu(m') \leftarrow ((t^\sigma, t), m)$; // az m' előzményének mentése
48 $\alpha(m') \leftarrow \alpha'$; // a változók aktuális értékének mentése

A [7] publikációban bemutattakhoz hasonlóan ennek a megközelítésnek az a lényege, hogy egy kibővített keresési térben folytassa a keresést az új optimális több perspektívás előtag-igazítás megtalálásához, amint egy új (c, a, u^n) esemény érkezik. Az algoritmus a (vezérlésfolyam perspektívában történő) kereséshez a c esetazonosítóhoz tárolt O nyitott és C zárt halmazt, valamint a κ_a tevékenységköltség-függvény felhasználásával kiterjesztett SPN-t alkalmazza. Annak érdekében, hogy más perspektívákat is figyelembe lehessen venni a keresés folyamán, a σ (kiterjesztett) nyomvonalat, a κ_v változóköltség-függvényt és az N_λ DPN folyamatmodell összetevőit is felhasználja. Továbbá, a számítási idő lerövidítése érdekében, minden esetben minden egyes állapothoz a jelenleg ismert optimális több perspektívás előtag-igazítást $(\overline{\gamma})$ és az aktuális változó értékeket (α) gyorsítótárba helyezi. Az OVAP-ok megoldásának gyorsítása érdekében az Ω OVAP gyorsítótárat használja.

Az algoritmus először a még fel nem fedezett állapotokat inicializálja (3–5. sorok), majd az O nyitott halmazban lévő állapotok elavult h értékeit frissíti (azaz a heurisztikus értékeket újraszámítja), így az f értékek is frissülnek (6–7. sorok). Ezután megkezdi az iteratív keresést az O nyitott halmazban (8. sor). Mindig a legkisebb f értékkel rendelkező m állapotot választja ki a nyitott halmazból (9. sor). Ha az m állapot célállapot (azaz a nyomvonalhálórész utolsó helyén van token) (10. sor), akkor megtalálta az optimális több perspektívás előtag-igazítást (azaz $\overline{\gamma}(m)$-t) és ezért visszatér a fő algoritmusba (11. sor). Ellenkező esetben az m állapotot áthelyezi a nyitott halmazból a zárt halmazba (12–13. sorok), és ellenőrzi, hogy lehetséges-e a közvetlen szinkronizálás (azaz egy helyes szinkronmozgás) végrehajtása (14–34. sorok). Ehhez először a változók új értékeit ellenőrzi és prímváltozóként beállítja azokat (18–23. sorok). Ha minden szükséges változó érvényes értékkel íródott, akkor az algoritmus továbblép az őrkifejezés kiértékelésére (24–26. sorok). Ha az őr kielégíthető (azaz a kiértékelt őrkifejezés logikai igaz értéket ad vissza), akkor lehetséges a közvetlen szinkronizálás. Ekkor végrehajtja azt (27–33. sorok), majd visszatér a fő algoritmusba, mivel megtalálta az optimális több perspektívás előtag-igazítást (34. sor).

Ha a közvetlen szinkronizálás nem lehetséges, a keresési folyamatot az aktuális m állapot utódállapotainak vizsgálatával folytatja (csak azokkal, amelyek nem elemei a C zárt halmaznak) (35–48. sorok). Minden egyes m' utódállapothoz a megfelelő γ_c vezérlésfolyam igazítást úgy kapja meg, hogy az előző m állapot több perspektívás előtag-igazításának a vezérlésfolyam-változatát kibővíti az m állapotból m' állapotba vezető SPN-átmenetet jelentő igazítási lépéssel (37. sor). Ezután a $getMPPA$ függvénnyel kiszámítja az előtag-igazítás több perspektívás változatát OVA-kkal (38. sor). Ha létezik OVA a γ_c igazításhoz, akkor egy több perspektívás előtag-igazítást ad vissza (γ'), ami lényegében a γ_c OVA-kkal kiegészítve. Ellenkező esetben egy üres sorozatot ad vissza. Egy állapot csak akkor kerül az O nyitott halmazba, ha létezik hozzá legalább egy érvényes több perspektívás előtag-igazítás, mivel csak azoknak az utódállapotai rendelkeznek érvényes több perspektívás igazítással. Ha az igazítás érvényes, és van már egy útvonal ugyanahhoz az m' állapothoz a nyitott halmazban, de az drágább, akkor a drágább útvonalat eltávolítja (40–41. sorok). Ezt követően az új (olcsóbb) útvonalat hozzáadja a nyitott halmazhoz (42–48. sorok).

2.3.4. Több perspektívás előtag-igazítás létrehozása

Ebben az alszakaszban az (OVA-val rendelkező) optimális több perspektívás előtag-igazítást előállító algoritmus (3. algoritmus) kerül bemutatásra. Egy vezérlésfolyam előtag-igazításhoz több OVA is létezhet, de az algoritmus csak egyet választ ki. Ha nincs OVA az adott előtag-igazításhoz, akkor egy üres sorozatot ad vissza.

Algoritmus 3: *getMPPA* (Optimális több perspektívás előtag-igazítás előállítása)

input : $N_\lambda = (P, T, F, V, val, in, wr, gd, [p_i], [p_f], \lambda_a, \lambda_v)$, $\sigma \in (\mathcal{A} \times \mathcal{U}^n)^\star$, $\gamma_c : \mathcal{R}(N^S, M_i^S) \to \Gamma_c^\star$, $\kappa_a : \Gamma_c \to \mathbb{R}_{\geq 0}$, $\kappa_v : V \times \mathcal{U} \times \mathcal{U} \to \mathbb{R}_{\geq 0}$, $\Omega : (T \times W)^\star \to W^\star \times \mathbb{R}_{\geq 0} \times W$

1 **begin**
2 $\overline{\rho} \leftarrow getVW(N_\lambda, \sigma, \gamma_c)$; // a változó írási szekvencia előállítása
3 $\omega \leftarrow \emptyset$; // az OVAP megoldást tároló változó értékének inicializálása
4 $cu = false$; // a gyorsítótár használatot jelző változó értékének inicializálása
5 **if** $\overline{\rho} \in dom(\Omega)$ **then** // ha az OVAP meg lett oldva korábban
6 $\omega, g_v, \alpha \leftarrow \Omega(\overline{\rho})$; // az OVAP megoldásának betöltése
7 $cu = true$;
8 **else** // ha az OVAP nem lett megoldva korábban
9 $VAR, CSTR, OBJ \leftarrow createOVAP(N_\lambda, \overline{\rho})$; // az OVAP létrehozása MILP problémaként
10 $\omega \leftarrow solveOVAP(VAR, CSTR, OBJ)$; // az OVAP megoldása
11 **if** $\omega = \emptyset$ **then** // ha nem létezik OVA a γ_c vezérlésfolyam igazításhoz
12 **return** $\langle\rangle, 0, \emptyset, \Omega$;
13 $\gamma \leftarrow \langle\rangle$; // a γ több perspektívás előtag-igazítás inicializálása
14 $g_a, i, j \leftarrow 0$; // az eddigi tevékenységköltség, valamint az i és j indexek inicializálása
15 **if** $cu = false$ **then** // ha az OVAP gyorsítótár nem lett használva
16 $g_v \leftarrow 0$; // az eddigi változóköltség inicializálása
17 $\alpha \leftarrow \bigcup_{v \in V} \{v \mapsto in(v)\}$; // változó értékek függvény inicializálása
18 **forall** $(a, t) \in \gamma_c$ **do** // minden vezérlésfolyam igazítási mozgás esetén
19 $g_a \leftarrow g_a + \kappa(a, t)$; // az igazítási mozgás tevékenységköltségének hozzáadása g_a-hoz
20 $s_l, s_m \leftarrow \gg$; // az eseménynapló lépés és a modell lépés inicializálása
21 **if** $a \neq \gg$ **then** // ha van eseménynapló lépés
22 $i \leftarrow i + 1$;
23 $u^n \leftarrow \pi_2(\pi_i(\sigma))$; // az attribútum értékek kinyerése a nyomvonalból
24 $s_l \leftarrow (a, u^n)$; // eseménynapló lépés kiegészítése attribútum értékekkel
25 **if** $t \neq \gg$ **then** // ha van modell lépés
26 $w \leftarrow \emptyset$; // a változó hozzárendeléseket tároló változó értékének inicializálása
27 **if** $wr(t) \neq \emptyset \vee gd(t) \neq$ `true` **then** // ha a t-hez van v. írási művelet vagy őrkifejezés
28 $j \leftarrow j + 1$;
29 $w \leftarrow \pi_j(\omega)$; // a modell lépéshez tartozó változó hozzárendelések kinyerése az ω OVAP megoldásból
30 **if** $cu = false$ **then** // ha az OVAP gyorsítótár nem lett haszálva
31 **forall** $v \in wr(t)$ **do** // minden t átmenet által írt változó esetén
32 $u \leftarrow \pi_2(\pi_j(\overline{\rho}))(v)$; // a v változóhoz rögzített (korrigált) érték kinyerése
33 $\alpha(v) \leftarrow w(v)$; // a v változó értékének frissítése a α-ban
34 $g_v \leftarrow g_v + \kappa_v(v, u, w(v))$; // a v változó költségének hozzáadása g_v-hez
35 $s_m \leftarrow (t, w)$; // modell lépés kiegészítése változó hozzárendelésekkel
36 $\gamma \leftarrow \gamma \cdot \langle(s_l, s_m)\rangle$; // a γ kibővítése a több perspektívás igazítási mozgással
37 **if** $cu = false$ **then** // ha az OVAP gyorsítótár nem lett haszálva
38 $\Omega(\overline{\rho}) \leftarrow (\omega, g_v, \alpha)$; // az OVAP megoldásának elmentése
39 $g \leftarrow g_a + g_v$; // a több perspektívás előtag-igazítás teljes költségének kiszámítása
40 **return** $\gamma, g, \alpha, \Omega$;

Az algoritmus három fő lépésből áll:

- Először a *getVW* függvény segítségével (4. algoritmus) **előállítja a $\overline{\rho}$ változó írási szekvenciát**, amivel azonosítható az az OVAP, amit az optimális több perspektívás előtag-igazítás előállításához meg kell oldani (2. sor).
- Ezt követően **megkeresi az OVAP megoldását (ω)** (3–12. sorok). Ha az OVAP meg lett oldva korábban (azaz $\overline{\rho}$ szerepel az Ω OVAP gyorsítótárban), akkor a megoldását a gyorsítótárból tölti be (5–6. sorok). Ellenkező esetben a *createOVAP* függvénnyel (5. algoritmus) létrehozza az OVAP-ot

(9. sor), majd a *solveOVAP* függvénnyel megoldja azt (10. sor). Ha nem létezik OVA az adott γ_c vezérlésfolyam igazításhoz, akkor (üres értékekkel) visszatér a $modA^{\star}_{inc}$ függvénybe (11–12. sorok).

- Következő lépésként **előállítja az optimális több perspektívás előtag-igazítást (γ)** a γ_c vezérlésfolyam igazítás, a σ nyomvonal, illetve az ω OVAP megoldás felhasználásával és **kiszámolja annak teljes költségét** (13–39. sorok). Egyúttal **a folyamatváltozók aktuális értékét (α) is megállapítja**.

Ha az algoritmus az Ω OVAP gyorsítótárat használni tudja, akkor az ω OVAP megoldás mellett az előállítandó több perspektívás előtag-igazításhoz tartozó változóköltséget (g_v-t) és a folyamatváltozók aktuális értékét (α-t) is megkapja a gyorsítótártól (6. sor), így azokat nem kell kiszámítania. Ha nem tudja használni a gyorsítótárat, akkor g_v-t és α-t az optimális több perspektívás előtag-igazítás előállítása során számítja ki (31–34. sorok). Továbbá, az előállítást követően az ω OVAP megoldást, és a kiszámított g_v-t és α-t (a $\overline{\rho}$ változó írási szekvenciához rendelve) elmenti OVAP gyorsítótárba (38. sor).

Változó írási szekvencia előállítása

A változó írási szekvencia előállítását a 4. algoritmus írja le. Az algoritmus bemenete a N_λ címkézett DPN folyamatmodell, a σ nyomvonal, valamint a γ_c vezérlésfolyam igazítás. Az algoritmus kimenete a $\overline{\rho}$ változó írási szekvencia.

Algoritmus 4: *getVW* (Változó írási szekvencia előállítása)

```
   input : N_λ = (P,T,F,V,val,in,wr,gd,[p_i],[p_f],λ_a,λ_v), σ ∈ (A × U^n)*, γ_c : R(N^S, M_i^S) → Γ_c^*
1  begin
2      ρ̄ ← ⟨⟩;                                          // a változó írási szekvencia inicializálása
3      i ← 0;                                           // index inicializálása a γ_c eseménynapló lépéseihez
4      forall (a,t) ∈ γ_c do                            // minden vezérlésfolyam igazítási mozgás esetén
5          if a ≠ ≫ then                                // ha van eseménynapló lépés
6              i ← i + 1;                               // index növelése
7          if t ≠ ≫ ∧ (wr(t) ≠ ∅ ∨ gd(t) ≠ true) then   // ha van modell lépés és a t átmenethez v. írási
             művelet vagy őrkifejezés tartozik
8              w ← ∅;                                   // a változó írások inicializálása
9              if a = λ_a(t) then                       // ha a mozgás szinkronmozgás
10                 forall v ∈ wr(t) do                  // minden t átmenet által írt v változó esetén
11                     u ← π_{λ_v(v)}(π_2(π_i(σ)));     // a v-hez rögzített érték kinyerése σ-ból
12                     if u ∈ val(v) then               // ha az u érték érvényes
13                         w ← w ∪ {v ↦ u};             // a v-hez u érték hozzárendelése
14                     else
15                         w ← w ∪ {v ↦ ∅};             // a v-hez ∅ (üres) érték hozzárendelése
16             else
17                 w ← ⋃_{v∈wr(t)} {v ↦ ∅};          // minden írandó v-hez ∅ (üres) érték hozzárendelése
18             ρ̄ ← ρ̄ · ⟨(t,w)⟩;                        // a modell lépés hozzáfűzése a ρ̄ tüzelési szekvenciához
19     return ρ̄;
```

Az algoritmus először inicializálja a $\overline{\rho}$ változó írási szekvenciát (2. sor) és az i indexet (3. sor), ami σ nyomvonalban való lépegetésre szolgál. Ezt követően minden olyan modell lépéshez, ami változó írási műveletekkel vagy őrkifejezéssel rendelkezik, összegyűjti a w változó írásokat, majd azokat a t átmenettel együtt a $\overline{\rho}$ változó írási szekvencia végéhez fűzi (7–18. sorok). Az olyan modell lépések, amikhez se változó írás, se őrkifejezés nem tartozik, az OVAP szempontjából lényegtelenek, így azokat nem kell a szekvenciához adni.

Szinkron mozgás esetén (9. sor) minden t átmenet által írt v változóhoz megnézi, hogy az u megfigyelt érték (azaz a változóhoz tartozó attribútum értéke a σ nyomvonalban) érvényes-e (10–12. sorok). Ha igen, akkor a megfigyelt értéket rendeli a változóhoz (13. sor), ha nem, akkor az üres értéknek megfelelő $\varnothing$ értéket (15. sor). Modell mozgás esetén minden írandó változóhoz az üres értéket rendeli (17. sor). Az olyan átmenetek esetében, amihez nem tartozik egy írási művelet sem, w üres halmaz marad.

OVAP létrehozása MILP-problémaként

Az OVAP létrehozását MILP-problémaként a 5. algoritmus írja le. Az algoritmus bemenete a N_λ címkézett DPN folyamatmodell és a $\overline{\rho}$ változó írási szekvencia. Az algoritmus kimenete az OVAP MILP probléma komponensei: a VAR MILP változók halmaza, a $CSTR$ MILP korlátozások halmaza, valamint az OBJ MILP objektív függvény.

Algoritmus 5: *createOVAP* (OVAP létrehozása MILP-problémaként)

```
   input  : N_λ = (P, T, F, V, val, in, wr, gd, [p_i], [p_f], λ_a, λ_v), ρ̄ ∈ (T × W)*
1  begin
2      VAR ← ⋃_{v∈V} (v_0);                                   // MILP változók inicializálása
3      CSTR ← ⋃_{v∈V} (v_0 = in(v));                          // a MILP kényszerek halmazának inicializálása
4      i ← 0;                                                 // a változók indexeinek inicializálása
5      forall (t, w) ∈ ρ̄ do                                  // minden modell lépés esetén
6          i ← i + 1;
7          forall v ∈ wr(t) do                                // minden t átmenet által írt v változó esetén
8              VAR ← VAR ∪ {v_i}; // MILP változó létrehozása a v i-edik modell lépéshez tartozó írási
                 műveletéhez
9              if w(v) ≠ ∅ then                               // ha a v-hez lett (érvényes) érték rögzítve
10                 VAR ← VAR ∪ {v̂_i};                         // megvalósíthatósági mutató létrehozása v_i-hez
11                 CSTR ← CSTR ∪ {v̂_i ⇔ (v_i = w(v))};        // MILP kényszerek létrehozása v írásához
12         if t ∈ dom(gd) then                                // ha a t átmenethez őrkifejezés tartozik
13             legyen MAP : V ∪ V' ↛ VAR;                     // MILP változó hozzárendelő függvény inicializálása
14             forall v ∈ wr(t) do                            // minden t átmenet által írt v változó esetén
15                 MAP ← MAP ⊕ {v' ↦ v_i};                    // MILP változó hozzárendelése v'-höz
16             forall v ∈ V do                                // minden v változó esetén
17                 i' ← max(i ∈ 0, ..., i − 1 | v_{i'} ∈ VAR);  // a legutóbbi változó írás megkeresése
18                 MAP ← MAP ⊕ {v ↦ v_{i'}};                  // MILP változó hozzárendelése v-hez
19             CSTR ← CSTR ∪ rewrite(gd(t), MAP);             // a t őrkifejezésének újraírása MILP
                 változókkal, MILP kényszerek létrehozása az őrkifejezéshez
20     OBJ ← Σ_{v̂_i ∈ VAR} (v̂_i κ_v(v));                      // objektív függvény létrehozása
21     return VAR, CSTR, OBJ;
```

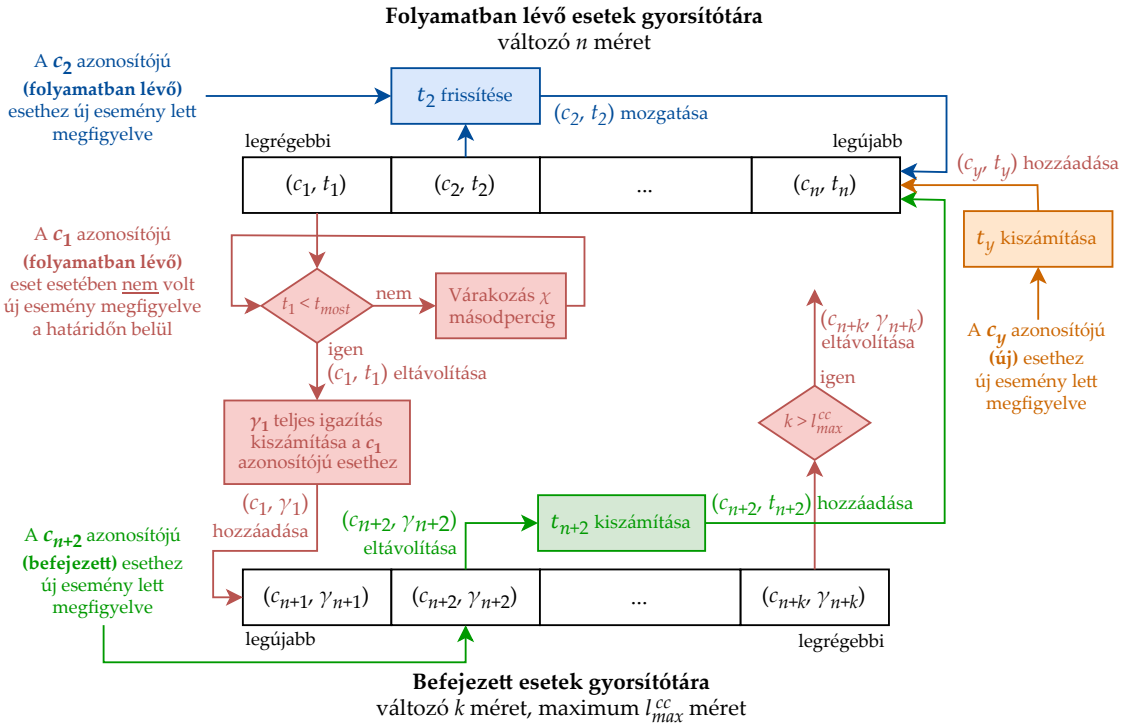
Az algoritmus először minden folyamatváltozó kezdeti értékéhez létrehoz egy-egy MILP változót (2. sor) és kényszert (3. sor), ezzel inicializálva a VAR MILP változók és a $CSTR$ MILP korlátozások halmazát. Következőként a MILP változókat és kényszereket az írási műveletekhez (7–11. sorok) és az őrkifejezésekhez (12–19. sorok) hozza létre, így létrehozva a MILP problémát. Legvégül a κ_v változóköltség-függvény segítségével létrehozza az OBJ objektív függvényt (20. sor).

Minden egyes **változó írási művelethez** egy v_i MILP változót hoz létre, ahol v a folyamatváltozót azonosítja, az i pedig azt a modell lépést, ahova az írási művelet tartozik (8. sor). Ha a változóhoz érvényes érték lett rögzítve (9. sor), akkor egy Boolean MILP változót is létrehoz (10. sor), ami egy megvalósíthatósági mutató a változóba írt értékéhez. Ez a mutató azt jelzi, hogy a nyomvonal szerint rögzített érték helyes-e vagy sem. Akkor vesz fel 1 értéket, ha sérti a DPN egy (vagy több) őrét, különben az értéke 0 lesz. Ezeknek a változóknak a (κ_v által meghatározott)

hibás változó hozzárendelés költségével megszorzott értékeinek az összege határozza meg az OBJ objektív függvény értékét (20. sor). Mivel a cél, hogy minél kevesebb különbség legyen a megfigyelt értékek és a folyamatmodell által választott értékek között, ez a függvény minimalizálásra kerül. Az elvárt egyenlőségeket MILP kényszerek formájában adja a problémához (11. sor).

Az algoritmus az **örökhöz** úgy hozza létre a MILP kényszereket, hogy az őrkifejezést a *rewrite* függvénnyel újra írja (19. sor). Ehhez minden v prímváltozóra való hivatkozást az aktuális v_i MILP változóra cserél (14–15. sorok), valamint minden v folyamatváltozóra való hivatkozást a legutóbbi változó írási műveletükhöz tartozó v_i MILP változóra cserél (16–18. sorok). Ezt követően hozza létre a MILP kényszereket (erről részletesen az [57] PhD dolgozatban lehet olvasni).

2.3.5. Az eset gyorsítótárazó módszer



2.3. ábra. A javasolt eset gyorsítótárazó módszer

A javasolt eset gyorsítótárazó módszer áttekintése a 2.3. ábrán látható. A módszer három paramétert használ:

- χ frissítési ráta (másodpercben megadva),
- $t_{\max}$ esetfrissítési határidő (másodpercben megadva),
- $l^{cc}_{\max}$ a befejezett esetek gyorsítótárának maximális mérete.

Az χ frissítési ráta határozza meg, hogy milyen időközönként kerülnek frissítésre az eset gyorsítótárak. A $t_{\max}$ esetfrissítési határidő értéke az eset legutolsó frissítési időpontjához adódik hozzá, és így kerül kiszámításra az eset következő frissítésének az elvárt időpontja. Ahogy az áttekintésben (2.3.1. alszakaszban) már említésre került, az alkalmazott eset gyorsítótárazó módszer az alábbi két gyorsítótárat veszi igénybe:

- **Folyamatban lévő esetek gyorsítótára**: Ez a gyorsítótár rugalmas és korlátlan méretű. Az esetekre vonatkozó bejegyzéseket növekvő sorrendben tárolja. Egy bejegyzés egy eset azonosítójából és egy időbélyegből álló rendezett kettes. Az időbélyeg az eset következő várható eseményének megfigyelési időpontját (azaz az eset elvárt frissítési időpontját) jelzi. A bejegyzések sorrendje az időbélyeg alapján van kialakítva úgy, hogy az első elem a legrégebben frissített esetre, az utolsó elem pedig a legutóbb frissített esetre vonatkozik.

- **Befejezett esetek gyorsítótára**: Ez a gyorsítótár rugalmas, de korlátozott (maximum $l^{cc}_{\max}$) méretű. Az esetekre vonatkozó bejegyzéseket a hozzáadás sorrendjében tárolja. Egy bejegyzés egy esetazonosítóból és az eset offline igazításából álló rendezett pár. Az offline igazítás az esetazonosító által meghatározott eset (feltételezhetően befejezett) nyomvonalából kerül kiszámításra.

A folyamatban lévő esetek gyorsítótárának mérete a $t_{\max}$ és a χ értékeitől függ. Minél nagyobb a $t_{\max}$, annál több eset halmozódhat fel a gyorsítótárban, mivel hosszabb ideig tartja az algoritmus őket folyamatban lévőként, még akkor is, ha valójában már befejeződtek. Ugyanakkor, ha a $t_{\max}$ túl kicsi, az esetek tévesen kerülhetnek át a befejezett esetek gyorsítótárába, holott még folyamatban vannak. A χ szintén hatással van a gyorsítótár méretére: ha az algoritmus gyakran vizsgálja az esetek állapotát (kisebb χ), a frissítési határidőt túllépő esetek kisebb ideig maradnak a gyorsítótárban, így a gyorsítótár mérete kisebb lesz. Nagyobb χ érték esetén pedig tovább maradnak benn, így a gyorsítótár mérete nagyobb lesz.

Folyamatban lévő esetek kezelése

A folyamatban lévő esetek gyorsítótárában egy esethez akkor jön létre vagy frissül a bejegyzés, ha egy új esemény kerül megfigyelésre hozzá. Az eset kezelése attól függ, hogy jelenleg milyen információval rendelkezünk róla. Ezek alapján az alábbi három eset lehetséges:

- egy új esethez új esemény került megfigyelésre,
- egy folyamatban lévő esethez új esemény került megfigyelésre,
- egy befejezett (azaz annak vélt) esethez új esemény került megfigyelésre.

A 2.3. ábrán az ilyen esetek kezelését (rendre) a sárga, a kék és a zöld színű folyamatábrák írják le. Mindhárom esetben az új, illetve a frissített bejegyzés a gyorsítótár végére kerül.

Ha **egy új esethez új esemény került megfigyelésre** (a példában ez a c_y azonosítójú eset), akkor az esethez kiszámításra kerül a következő frissítés elvárt időpontja (t_y), majd az esetazonosítót és az időbélyeget tartalmazó bejegyzés hozzáadódik a gyorsítótár végéhez.

Ha **egy folyamatban lévő esethez új esemény került megfigyelésre** (a példában ez a c_2 azonosítójú eset), akkor az esethez újra kiszámításra kerül a következő frissítés elvárt időpontja (t_2), majd az esetazonosítót és az időbélyeget tartalmazó bejegyzés áthelyeződik a gyorsítótár végéhez.

Ha **egy befejezettnek vélt esethez új esemény került megfigyelésre** (a példában ez a c_{n+2} azonosítójú eset), akkor az esethez tartozó bejegyzés törlődik a befejezett esetek gyorsítótárából. Ezt követően az esethez kiszámításra kerül a következő frissítés elvárt időpontja (t_{n+2}), majd az esetazonosítót és az időbélyeget tartalmazó bejegyzés hozzáadódik a folyamatban lévő esetek gyorsítótárának a végéhez.

Befejezett esetek kezelése

A gyorsítótárak használata már a folyamat megfigyelésének az elejétől megkezdődik. Amikor már van legalább egy folyamatban lévő eset (tehát a folyamatban lévő esetek gyorsítótára nem üres), akkor az algoritmus megadott időközönként ellenőrzi, hogy a tároló elején lévő eset (azaz a legrégebben frissített eset) túllépte-e a megadott időkorlátot (azaz az esetazonosítóval rögzített várható frissítési időpont már elmúlt). A 2.3. ábrán az ilyen esetek kezelését a piros színnel jelölt folyamatábra mutatja be.

Az ábrán látható példában a c_1 azonosítójú eset a legrégebben frissült folyamatban lévő eset, ezért az algoritmus megnézi, hogy a t_1 kisebb-e mint t_{most} (azaz az aktuális időpont). Ha nem kisebb, akkor vár χ másodpercet, majd újra megvizsgálja. Ha kisebb, akkor eltávolítja az esethez kapcsolódó bejegyzést a folyamatban levő esetek gyorsítótárából, kiszámítja az esethez az offline igazítást (γ_1), majd az esetazonosítót és az igazítást tartalmazó rendezett párost (a példa szerint (c_1, γ_1)-t) hozzáadja a befejezett esetek gyorsítótárának az elejéhez.

Ha a befejezett esetek gyorsítótárának a mérete (k) túllépi a megadott maximális méretet ($l^{cc}_{\max}$), akkor a gyorsítótár utolsó bejegyzése törlésre kerül. Ekkor a törölt bejegyzésben lévő esetazonosítóhoz tartozó esetet véglegesen befejezettnek tekintjük. Ennek következtében nem csak a befejezett esetek gyorsítótárából törlődnek az esethez tartozó objektumok, hanem a többi gyorsítótárból is.

2.4. A javasolt folyamatadat vizualizációs módszerek

Ebben a szakaszban az általam javasolt folyamatadat vizualizációs módszerek részletes bemutatására kerül sor, amelyek az eredeti eseményadatok és az igazítás-alapú megfelelőség ellenőrzési eredmények megjelenítésére szolgálnak. Először a közös jellemzők, majd az egyedi tulajdonságok kerülnek kifejtésre.

A jobb megértés érdekében mindkét módszer egy-egy konkrét példával van illusztrálva. A példákban egy automatizált összeszerelési folyamat nyolc egymást követő műveletből álló („t_{s1o}”, „t_{s2o}” stb. feliratú) nyomvonalára történik utalás. Ebben a folyamatban a hibák (a „t_{err}” címkével) a folyamat bármelyik szakaszában jelentkezhetnek. A folyamat részletesebb leírása a C. függelék C.1.1. alszakaszában olvasható.

2.4.1. Áttekintés

A vizualizációs módszereket Tamara Munzner vizuális analitikai keretrendszere alapján fejlesztettem ki, amely a vizualizációval kapcsolatban három kulcsfontosságú szempontot vizsgál: „Mit?”, „Miért?” és „Hogyan?” [112]. A 2.4. ábrán a kifejlesztett két folyamatadat vizualizációs módszer összehasonlítása látható egyszerűsített, táblázatos formában.

A „Mit?” szempont a vizualizációhoz rendelkezésre álló adatokra vonatkozik. Az én esetemben az adatforrások közé az eseményfolyamból kinyert eseményadatok, valamint a javasolt MOCC módszer kimenetei (azaz igazítási adatok) tartoznak. Az eseményadatok egyszerű vagy összetett eseményfolyam megfigyelt egységeiből származnak. Ennek megfelelően, az adatokkal szemben támasztott tartalmi elvárások

is két csoportba oszthatók. (A 2.4. ábrán a csillaggal jelölt attribútumok megléte csak egyszerű eseményfolyamból származó eseményadatok esetében elvárás.) Az igazítási adatok az eseményadatokból és az előre meghatározott folyamatmodellből kiszámított igazításokból származnak. Egy adategység egy több perspektívás (előtag-)igazítási mozgás, aminek négy típusa lehet.

A „Miért?” szempont a vizualizáció fő célját, valamint a lehetséges mellékcélokat határozza meg. Az én esetemben mindkét vizualizáció elsődleges célja, hogy segítse a folyamatok végrehajtásának nyomon követését. Az eseményadatok vizualizációjának az a további célja, hogy átfogó képet nyújtson a valós folyamatról, az igazítási adatok vizualizációjának pedig az, hogy kimutassa az elvárt folyamattól való eltéréseket. Együttes alkalmazásuk esetén elősegítik az eltérések gyors észlelését és a kiváltó okok beazonosítását, lehetővé téve a felhasználók számára a korrekciós intézkedések mihamarabbi végrehajtását.

A „Hogyan?” szempont az adatok megjelenítésére vonatkozó tervezési döntéseket határozza meg. Az irodalmi áttekintésből származó meglátások alapján olyan tervezési döntéseket kívántam meghozni, amelyek magukban foglalják a meglévő vizualizációs módszerek előnyeit, ugyanakkor kezelik azok korlátjait. A részletes leírás a további alszakaszokban olvasható.

2.4. ábra. Összefoglaló táblázat a javasolt folyamatadat vizualizációs módszerekről

2.4.2. A módszerek közös tulajdonságai

Mind az eseményadatokat, mind az igazítási adatokat egy Gantt-diagram által inspirált vegyes (fix és időtartamos) idővonal-alapú vizualizációval jelenítem meg. A vizualizáció egyik fő kihívása az időbeni átfedő események megjelenítése, amire számos módszer jelent már meg [113]. Tekintettel a többszörös átfedés előfordulásának lehetőségére, ezek közül két technika tűnik megfelelőnek: az események párhuzamos függőleges felsorolása (Gantt-diagramhoz hasonlóan) azzal a két választási lehetőség-

gel, hogy az események részben vagy egyáltalán nem fedik egymást. Bár az átfedések tömöríthetik a diagram méretét, akadályozhatják az áttekinthetőséget. Emiatt a nem átfedő vizuális elemek mellett döntöttem.

Az y-tengelyen az események egyik (nem időértéket tartalmazó) attribútuma kerül alkalmazásra. Ha az erőforrás meg van adva, akkor az lesz az alapértelmezett attribútum; ha nincs, akkor az eset azonosítója kerül felhasználásra. Fontos, hogy a felhasználók csak olyan attribútumot választhatnak ki, amely minden eseményben szerepel. Az objektumok átfedés nélküli megjelenítéséhez szükséges helytől függően egy attribútumérték több sorban is megjelenítésre kerülhet. Többsoros megjelenítés esetén az elsődleges cél az, hogy a sorok száma minimális legyen. Ezzel a diagram magassága minimális mértéken tartható, ami az áttekinthetőség miatt fontos. A másodlagos cél pedig az, hogy az azonos esethez tartozó objektumok minél kevesebb sorban legyenek szétszórva. Ez a folyamatvégrehajtások követhetősége miatt fontos.

A megjelenített objektumokat (azaz eseményeket és igazítási mozgásokat) kis függőleges vonalak („|”) jelölik. Az azonos tevékenység-példányhoz tartozó eseményeket és igazítási mozgásokat vastagabb vonalak (sávok) kötik össze. Egy egyszerű eseményfolyam megfigyelt egysége egyetlen eseménynek tekintendő, míg egy összetett eseményfolyam megfigyelt egységét két különálló eseményként kell kezelni (kezdő és befejező esemény).

Az interaktivitás növelése érdekében a felhasználók egy attribútum értéke alapján szűrhetik a diagrammon megjelenített objektumokat. Ekkor csak azok az események és igazítási mozgások jelennek meg, amelyekben a kiválasztott attribútum a kiválasztott értékkel rendelkezik. Ez ideális például bizonyos erőforrások teljesítményének elemzésére. További interaktivitást nyújt az, hogy az objektumok fölé mozgatva az egeret a releváns információk eszköztippekben megjelennek, valamint a kapcsolódó objektumok az esetazonosítójuk alapján, megfigyelési sorrendben összekötésre kerülnek. Ezek az interaktív funkciók mélyebb betekintést biztosítanak a felhasználóknak anélkül, hogy túlzsúfolnák a képernyőt.

A két megjelenítési módszer közötti különbségek elsősorban a színválasztásban és az eszköztippek tartalmában mutatkoznak meg. Ezek a különbségek a következő alszakaszokban kerülnek részletezésre.

2.4.3. Az eseményadatok vizualizálása

Az eseményadatok vizualizálásakor az eseményeket és azok kapcsolatait ábrázoló objektumok a tevékenység attribútum értéke alapján vannak színezve. A színek egy előre meghatározott kategorikus színskálából rendelődnek az értékekhez, az (új) értékek megfigyelésének sorrendjében.

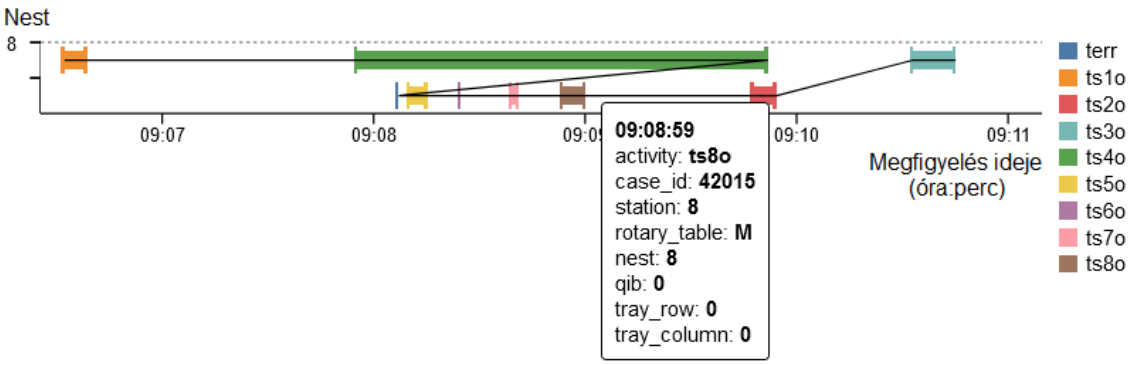
Ha a felhasználó az egeret az eseményt ábrázoló objektum fölé viszi, akkor egy eszköztippben megjelenik a megfigyelt esemény összes eredeti információja, attribútumnév és -érték párokként, sorokba rendezve. Az időpont elsőbbséget élvezve az első sorban kerül megjelenítésre, de csak az időérték része, mivel a dátum az többnyire azonos egy megfigyelési munkamenet során. Egy összetett eseményfolyam megfigyelt egységei esetében a kezdő esemény objektum eszköztippje a tevékenységpéldány kezdő időbélyegét, míg a befejező esemény objektum annak befejező időbélyegét mutatja.

Következőként egy példa kerül bemutatásra az eseményadatok vizualizálására, amihez egy összeszerelési folyamat összetett eseményfolyamából származó, egy adott

esetére vonatkozó eseményadatok kerülnek felhasználásra. A 2.5. ábrán a bemeneti eseményadatok, a 2.6. ábrán pedig az eseményadatok vizualizációja látható, ahol a „nest” attribútum van az y-tengelyen megjelenítve. A kurzor a „t_{s8o}” feliratú befejező esemény objektum fölött van, amely a 8. állomáson végzett műveleteket jelöli. Már a megfelelőség ellenőrzés elvégzése előtt fedezhető fel eltérés az elvárt viselkedéstől. Az események sorrendje helytelen (például a „t_{s1o}” után a „t_{s2o}” helyett a „t_{s4o}” következik), és a „t_{s4o}” tevékenység időtartama szokatlanul hosszú. Továbbá, mivel a termék az 5. állomás elérése előtt hibásnak lett minősítve, azt a 7. állomásnak ki kellett volna dobnia. A termék azonban tovább haladt a 8. állomásig.

```
{ id: 171, values: {case_id: 42015, activity: 'ts1o', start_timestamp: new Date("2016-11-11 09:06:31.812"),
complete_timestamp: new Date("2016-11-11 09:06:38.278"), station: 1, rotary_table: 'M', nest: 8, qib: 1} },
{ id: 176, values: {case_id: 42015, activity: 'ts4o', start_timestamp: new Date("2016-11-11 09:07:54.968"),
complete_timestamp: new Date("2016-11-11 09:09:51.429"), station: 4, rotary_table: 'M', nest: 8, qib: 0} },
{ id: 178, values: {case_id: 42015, activity: 'terr', start_timestamp: new Date("2016-11-11 09:08:06.614"),
complete_timestamp: new Date("2016-11-11 09:08:06.614"), station: 4, rotary_table: 'M', nest: 8,
error_code: 407} },
{ id: 179, values: {case_id: 42015, activity: 'ts5o', start_timestamp: new Date("2016-11-11 09:08:09.765"),
complete_timestamp: new Date("2016-11-11 09:08:14.778"), station: 5, rotary_table: 'M', nest: 8, qib: 0} },
{ id: 181, values: {case_id: 42015, activity: 'ts6o', start_timestamp: new Date("2016-11-11 09:08:24.171"),
complete_timestamp: new Date("2016-11-11 09:08:24.171"), station: 6, rotary_table: 'M', nest: 8, qib: 0} },
{ id: 183, values: {case_id: 42015, activity: 'ts7o', start_timestamp: new Date("2016-11-11 09:08:38.718"),
complete_timestamp: new Date("2016-11-11 09:08:40.641"), station: 7, rotary_table: 'M', nest: 8, qib: 0} },
{ id: 184, values: {case_id: 42015, activity: 'ts8o', start_timestamp: new Date("2016-11-11 09:08:53.234"),
complete_timestamp: new Date("2016-11-11 09:08:59.674"), station: 8, rotary_table: 'M', nest: 8, qib: 0,
tray_row: 0, tray_column: 0} },
{ id: 185, values: {case_id: 42015, activity: 'ts2o', start_timestamp: new Date("2016-11-11 09:09:47.281"),
complete_timestamp: new Date("2016-11-11 09:09:53.788"), station: 2, rotary_table: 'M', nest: 8, qib: 1} },
{ id: 186, values: {case_id: 42015, activity: 'ts3o', start_timestamp: new Date("2016-11-11 09:10:32.562"),
complete_timestamp: new Date("2016-11-11 09:10:44.627"), station: 3, rotary_table: 'M', nest: 8, qib: 1} }
```

2.5. ábra. Egy összeszerelési folyamat összetett eseményfolyamából származó, egy adott esetére vonatkozó eseményadatok



2.6. ábra. Egy összeszerelési folyamat összetett eseményfolyamából származó, egy adott esetére vonatkozó eseményadatok vizualizációja

2.4.4. Az igazítási adatok vizualizálása

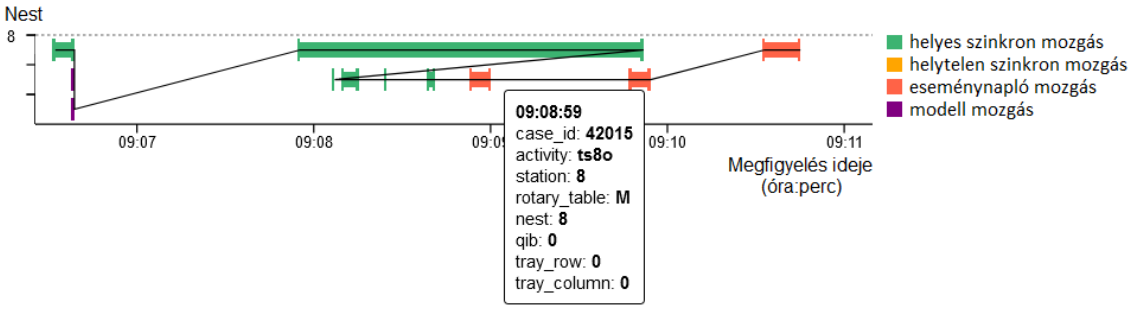
Az igazítási adatok – konkrétan a több perspektívás előtag-igazítási mozgások – vizualizálásában az igazítási mozgásokat és azok kapcsolatait ábrázoló objektumok az igazítási mozgás típusa alapján vannak színezve. Az igazítási mozgás típusa egy olyan attribútum, amelynek az értékét az MOCC módszer határozza meg. Az egyszerűség kedvéért ezek az értékek számértékként kerülnek eltárolásra és a színek egy előre meghatározott kategorikus színskálából rendelődnek hozzájuk. A lehetséges számértékek, igazítási mozgás típusok és a hozzájuk tartozó színek a következők:

1. helyes szinkron mozgás: zöld,
2. helytelen szinkron mozgás: sárga,
3. eseménynapló mozgás: piros,
4. modell mozgás: lila.

Ha a felhasználó az egeret az igazítási mozgást ábrázoló objektum fölé viszi, akkor egy eszköztippben megjelenik a kapcsolódó megfigyelt eseményhez tartozó összes eredeti információ. Ha van a folyamatmodell által javasolt javított érték egy attribútumhoz, akkor az eszköztippben az adott attribútumhoz tartozó sor piros színnel kerül kiemelésre, és a javított érték az eredeti érték mellé fűzve kerül megjelenítésre.

```
{ id: 208, move_type: 1, values: {case_id: 42015, activity: 'ts1o', start_timestamp: new Date("2016-11-11 09:06:31.812000"),
complete_timestamp: new Date("2016-11-11 09:06:38.278000"), station: 1, rotary_table: 'M', nest: 8, qib: 1.0} },
{ id: 209, move_type: 4, values: {case_id: 42015, activity: 'ts2o', qib: 1, station: 2, start_timestamp: new Date("2016-11-11
09:06:38.278000"), complete_timestamp: new Date("2016-11-11 09:06:38.278000"), rotary_table: 'M', nest: 8} },
{ id: 210, move_type: 4, values: {case_id: 42015, activity: 'ts3o', qib: 1, station: 3, start_timestamp: new Date("2016-11-11
09:06:38.278000"), complete_timestamp: new Date("2016-11-11 09:06:38.278000"), rotary_table: 'M', nest: 8} },
{ id: 211, move_type: 1, values: {case_id: 42015, activity: 'ts4o', start_timestamp: new Date("2016-11-11 09:07:54.968000"),
complete_timestamp: new Date("2016-11-11 09:09:51.429000"), station: 4, rotary_table: 'M', nest: 8, qib: 0.0} },
{ id: 212, move_type: 1, values: {case_id: 42015, activity: 'terr', start_timestamp: new Date("2016-11-11 09:08:06.614000"),
complete_timestamp: new Date("2016-11-11 09:08:06.614000"), station: 4, rotary_table: 'M', nest: 8, error_code: 407.0} },
{ id: 213, move_type: 1, values: {case_id: 42015, activity: 'ts5o', start_timestamp: new Date("2016-11-11 09:08:09.765000"),
complete_timestamp: new Date("2016-11-11 09:08:14.778000"), station: 5, rotary_table: 'M', nest: 8, qib: 0.0} },
{ id: 214, move_type: 1, values: {case_id: 42015, activity: 'ts6o', start_timestamp: new Date("2016-11-11 09:08:24.171000"),
complete_timestamp: new Date("2016-11-11 09:08:24.171000"), station: 6, rotary_table: 'M', nest: 8, qib: 0.0} },
{ id: 215, move_type: 1, values: {case_id: 42015, activity: 'ts7o', start_timestamp: new Date("2016-11-11 09:08:38.718000"),
complete_timestamp: new Date("2016-11-11 09:08:40.641000"), station: 7, rotary_table: 'M', nest: 8, qib: 0.0} },
{ id: 216, move_type: 3, values: {case_id: 42015, activity: 'ts8o', start_timestamp: new Date("2016-11-11 09:08:53.234000"),
complete_timestamp: new Date("2016-11-11 09:08:59.674000"), station: 8, rotary_table: 'M', nest: 8, qib: 0.0,
tray_row: 0.0, tray_column: 0.0} },
{ id: 217, move_type: 3, values: {case_id: 42015, activity: 'ts2o', start_timestamp: new Date("2016-11-11 09:09:47.281000"),
complete_timestamp: new Date("2016-11-11 09:09:53.788000"), station: 2, rotary_table: 'M', nest: 8, qib: 1.0} },
{ id: 218, move_type: 3, values: {case_id: 42015, activity: 'ts3o', start_timestamp: new Date("2016-11-11 09:10:32.562000"),
complete_timestamp: new Date("2016-11-11 09:10:44.627000"), station: 3, rotary_table: 'M', nest: 8, qib: 1.0} }
```

2.7. ábra. Egy összeszerelési folyamat összetett eseményfolyamából származó, egy adott esetére vonatkozó eseményadatokból számított igazítási adatok



2.8. ábra. Egy összeszerelési folyamat összetett eseményfolyamából származó, egy adott esetére vonatkozó eseményadatokból számított igazítási adatok vizualizációja

Minden igazítási mozgás a kapcsolódó megfigyelt esemény eredeti rögzített értékeinek felhasználásával kerül megjelenítésre, még akkor is, ha valamelyiket a folyamatmodell hibásnak ítélt meg. Ez alól a modell mozgások kivételt képeznek, mivel egy modell mozgáshoz nem tartozik megfigyelt esemény, így az attribútumoknak sincsen eredeti értékük. Ebből adódóan, a modell mozgások csak a modell által ajánlott időértékkel ábrázolhatók. Ha a modell nem ad meg időbeli információt egy tevékenységre vonatkozóan, akkor ez az időérték az előző igazítási mozgás (befejezési) időbélyege, vagy ha az nem áll rendelkezésre, akkor a következő igazítási mozgás (kezdő) időbélyege. Figyelembe véve, hogy az igazítás kiszámítása egy esetre vonatkozóan csak akkor történik meg, ha legalább egy megfigyelt esemény tartozik hozzá, ez a megközelítés biztosítja, hogy a folyamatmozgásokhoz mindig rendelkezésre álljon egy időérték. Ugyanez a megközelítés van alkalmazva olyan esetekben

is, amikor a modell nem határoz meg (konkrét) értéket az y-tengelyen megjelenített attribútumhoz.

Következőként egy példa kerül bemutatásra az igazítási adatok vizualizálására, amihez egy összeszerelési folyamat összetett eseményfolyamából származó, egy adott esetére vonatkozó eseményadatokból számított igazítási adatok kerülnek felhasználásra. A 2.7. ábrán a bemeneti igazítási adatok, a 2.8. ábrán pedig az igazítási adatok vizualizációja látható, ahol a „nest" attribútum van az y-tengelyen megjelenítve. A kurzor a „t_{s8o}" feliratú igazítási mozgás befejező objektuma fölött van. Ez egy eseménynapló mozgás, mivel a modell szerint a terméket a 7. állomáson el kellett volna dobni. (Azaz a terméknek nem lett volna szabad eljutnia a 8. állomásig.) Megjegyzendő, hogy a tálca sor- és oszloppozícióját jelző számok (ahová a késztermék kerül) nulla értékűek. Ez azt jelentheti, hogy a termék vagy nem került a tálcára, vagy hiba történt e számok észlelése vagy rögzítése során. A nagyobb jelentőségű probléma az, hogy a „t_{s2o}" és a „t_{s3o}" időbeli adatai helytelenek (amint az a 2.6. ábrán is látható). Ezeket az eseményeket azután észlelték, hogy a termék elérte az utolsó állomást, nem pedig a „t_{s1o}" után. Emiatt van két modell mozgás a „t_{s1o}" után, valamint két eseménynapló mozgás az igazítás végén.

2.5. A javasolt megoldások hatékonyságának vizsgálata

Az ebben a fejezetben bemutatott megoldásokhoz kapcsolódóan a következő vizsgálatok lettek elvégezve:

- **MOCC vizsgálatok** (2.5.1. alszakasz): A vizsgálatok célja a javasolt MOCC módszer hatékonyságának tesztelése. Mivel nincs más MOCC megoldás, ezért kétfajta vizsgálatot végeztem:
  - **OCC vizsgálat**: A vizsgálat célja az MOCC módszer futási idejének mérése és összehasonlítása egy OCC módszerével. A vizsgálat keretében mindkét módszert egy valós folyamatra alkalmaztam.
  - **MCC vizsgálat**: A vizsgálat célja az MOCC módszer kimenetek (azaz a több perspektívás előtag-igazítások) minőségének értékelése és összehasonlítása egy MCC módszerével. A vizsgálat keretében mindkét módszert három valós folyamatra alkalmaztam.

- **Folyamatadat vizualizációs vizsgálat** (2.5.2. alszakasz): A vizsgálat célja a javasolt folyamatadat (eseményadat és igazítási adat) vizualizációs módszerek használhatóságának tesztelése. A vizsgálathoz egy valós folyamat lett felhasználva.

Az OCC vizsgálathoz a vezérlésfolyam (előtag-)igazítást számító OCC megoldást [7] alkalmaztam, az MCC vizsgálathoz pedig a kiegyensúlyozott MCC (röviden BMCC) megoldást [13]. A vizsgálatokhoz összesen három valós folyamat folyamatmodellje és folyamatadatai lettek felhasználva:

- egy automatizált tekercs összeszerelési folyamat,
- egy közúti közlekedési bírságkezelési folyamat, és
- egy kórházi számlázási folyamat.

Ezek közül csak az első folyamat rendelkezik rövid átfutási idővel (normál esetben kb. 1–2 perc), így csak azt tudtam az OCC vizsgálathoz és a folyamatadat vizualizációs vizsgálathoz felhasználni. Az MCC vizsgálathoz mindhárom folyamatot felhasználtam. Fontos, hogy az utolsó kettő folyamatot már vizsgálták korábban a BMCC megoldással [57], mert így a megoldástól elvárt kimenetek ismertek. A felhasznált folyamatok a C. függelék C.1. szakaszában kerülnek részletes bemutatásra.

Megvalósítások

A DPN folyamatmodelleket a ProM (6.9) szoftver [114] „Create DPN (Text language based)" nevű bővítménye segítségével hoztam létre, majd PNML fájlba exportáltam és minden folyamatváltozóhoz hozzáadtam egy kezdeti értéket. Az automatizált tekercs összeszerelési folyamathoz szükség volt az eseménynaplók átalakítására, amihez MySQL Server-t és Workbench-et használtam.

Az MOCC módszer Python (3.11) nyelven implementáltam, a PM4Py (2.7.5.2) könyvtár [115] bővítéseként. (A PM4Py egy nyílt forráskódú folyamatbányászati programkönyvtár Python nyelven.) Az OVAP MILP-problémaként való megoldásához a Google OR-Tools (9.6) könyvtárat [116] használtam. Az **OCC vizsgálat** elvégzéséhez a választott OCC megoldás eredeti implementációját [117] – ami a PM4Py egy korábbi verziójának bővítése – átraktam ugyanabba a PM4Py verzióba, amibe az MOCC módszer implementációját is raktam. Ezen felül a PM4Py könyvtárat DPN beolvasó, valamint OVAP-ot generáló és megoldó kóddal is kiegészítettem. Az **MCC vizsgálat** elvégzéséhez a BMCC megoldást a ProM (6.9) szoftver „DataAwareReplayer" nevű csomagjának „Conformance Checking of DPN" nevű bővítményének használatával alkalmaztam. A PM4Py módosított verziója, a vizsgálatokhoz felhasznált DPN folyamatmodellek és eseménynaplók, valamint a vizsgálatokat végrehajtó kódok teljes egészében elérhetők a GitHub-on[1].

A **folyamatadat vizualizációs vizsgálat**hez egy tesztkörnyezetet hoztam létre webalkalmazás formájában. A backend egy szerverből és egy middleware-ből, a frontend pedig egy kliensből áll. Ezek a következő feladatokat látják el:

- A szerver fő feladata az eseményfolyam szimulálása a megadott eseménynapló és a beállítások alapján. A beállítások tartalmazzák a (csv formátumú) eseménynapló és a (pnml formátumú) folyamatmodell elérési útvonalát, az eseményfolyam típusát (egyszerű eseményfolyam vagy összetett eseményfolyam), a kiválasztott eseményfolyam-típus minden egyes szükséges attribútumának attribútumértékét tartalmazó oszlop számát, az egyes attribútumok értéktípusát és az időbélyegek formátumát.

- A middleware figyeli a kiszolgáló által szolgáltatott eseményadatokat, majd feldolgozza azokat, és elemzést végez rajtuk. Végül elküldi a kimeneteket a csatlakozott klienseknek. Amikor egy új kliens csatlakozik, a middleware elküldi a vizualizációhoz szükséges összes információt és a vizualizációhoz szükséges összes gyorsítótárazott adatot.

- A vizualizációkat a middleware által szolgáltatott adatok és a felhasználó által megadott beállítások alapján a kliens oldalán generálja és jeleníti meg. A két vizualizáció egyetlen oldalon jelenik meg, lehetővé téve a felhasználó számára,

[1] `https://github.com/zsuzsanna-nagy/mocc`

hogy mindkettőt egyszerre vizsgálja. A felhasználó mindkét vizualizációban egyidejűleg nagyíthat, kicsinyíthet, visszaállíthatja a nagyítást, görgethet egy kicsit balra vagy jobbra.

A szervert és a middleware-t Python nyelven implementáltam. A middleware a PM4Py könyvtár módosított változatát (lásd előző bekezdés) és a Flask (2.3.3) keretrendszert használja a Flask-SocketIO (5.3.6) könyvtárral együtt. A klienst webes technológiákkal (HTML, CSS és JavaScript) valósítottam meg. A kliens a D3 (vagy D3.js) JavaScript könyvtárat használja a vizualizációk létrehozására és frissítésére, a Socket.IO JavaScript könyvtárat pedig a middleware-rel történő kommunikációra. A szerver TCP socketeken keresztül szerializált JSON-üzenetek formájában kommunikál a middleware-rel az adatok továbbításához. A middleware JSON-üzenetek formájában valós időben küldi az adatokat a kliensnek a Flask-SocketIO által támogatott WebSockets segítségével. A tesztkörnyezetet megvalósító kódok a bemeneti fájlokkal együtt elérhetők a GitHub-on[2].

Mindegyik vizsgálatot Windows 10 operációs rendszerű, Intel(R) Core(TM) i5-3320M 2,60 GHz-es, kétmagos processzorral és 8 GB RAM-mal felszerelt laptopon végeztem.

2.5.1. A javasolt MOCC módszer hatékonyságának vizsgálata

Vizsgálati környezet és paraméterek

A javasolt MOCC módszer teszteléséhez mindhárom folyamat (az automatizált tekercs összeszerelési folyamat, a közúti közlekedési bírságkezelési folyamat, valamint a kórházi számlázási folyamat) folyamatmodelljét és az eseményadataiból generált naplófájlokat felhasználtam. Az eredeti eseménynaplók felhasználásával mindhárom folyamathoz kétféle eseménynaplót hoztam létre:

- Csak a folyamatmodellre tökéletesen **illeszkedő nyomvonalak**at tartalmazó eseménynapló: Ez az eseménynapló olyan nyomvonalakat tartalmaz, amiknek a megfelelőségi értéke pontosan 1 és az optimális több perspektívás igazításuk teljes költsége 0.
- Csak a folyamatmodellre **nem teljesen illeszkedő nyomvonalak**at tartalmazó eseménynapló: Ez az eseménynapló olyan nyomvonalakat tartalmaz, amiknek a megfelelőségi értéke 1-nél kisebb, és az optimális több perspektívás igazításuk teljes költsége 0-nál nagyobb.

A tekercs összeszerelési folyamat esetében, minden nyomvonalhoz több esetet is hozzáadtam a naplófájlokhoz. (Különböző eseteknek lehet ugyanaz a nyomvonala.) Így 67 olyan esetet gyűjtöttem össze, amelyeknek a nyomvonala teljesen illeszkedik, és 85 olyan esetet, amelyeknek a nyomvonala nem teljesen illeszkedik a folyamatmodellre. A másik két folyamat esetében, mivel azoknál túl sokféle nyomvonal van, csak a 10 leggyakoribb illeszkedő, illetve a 10 leggyakoribb nem teljesen illeszkedő nyomvonalat vettem figyelembe. Mindegyik nyomvonalhoz 5 esetet választottam ki, így a naplófájlok összesen 50–50 esetet tartalmaznak.

Az igazítások számításához többnyire az alapértelmezett költségfüggvényeket (tevékenységköltség-függvényt és változóköltség-függvényt) alkalmaztam, így az eltérések költsége egységesen 1 volt minden perspektívában. Ez alól kivételt csak az

[2] `https://github.com/zsuzsanna-nagy/process_data_visualization`

összeszerelési folyamat képezett, ahol a használt változóköltség-függvényben az eltérések költsége 1 helyett 0,1 volt. Ezzel azt kívántam kifejezni, hogy a vezérlésfolyam perspektívában jelentkező eltérések nagyobb súllyal bírnak, vagyis nagyobb a valószínűsége annak, hogy a többi perspektívához kapcsolódó rögzített adat helytelen, mint annak, hogy a végrehajtott állomásműveletek sorrendjében van eltérés (ami fizikailag nem lehetséges).

Az **OCC vizsgálat** során az OCC megoldást a tekercs összeszerelési folyamat DPN folyamatmodelljének vezérlésfolyam másolatára (azaz egyszerű Petri-háló változatára) alkalmaztam. A két megoldás számítási időigényét összehasonlítottam egymással. Várhatóan az MOCC megoldás lassabb, de azért aránylag gyors (pl. esetenként kevesebb, mint 1 másodperc a számítási idő).

Az **MCC vizsgálat** esetében elvárás, hogy azonos nyomvonalak esetén az MOCC megoldás ugyanazokat, vagy legalább azonos költségű igazításokat adjon eredményül, mint a BMCC megoldás. Ez alól kivételt csak a hiányos nyomvonalak képezhetnek, mivel online környezetben a nem teljes nyomvonalak nem tekinthetők helytelen viselkedésnek, ha a folyamat még nem fejeződött be. A vizsgálat során azonban az MOCC megoldást az eset gyorsítótárazási módszerrel kiegészítve alkalmaztam, így az igazítás számítása során minden folyamatlefutást befejezettnek tekintett (azaz offline igazításokat számolt).

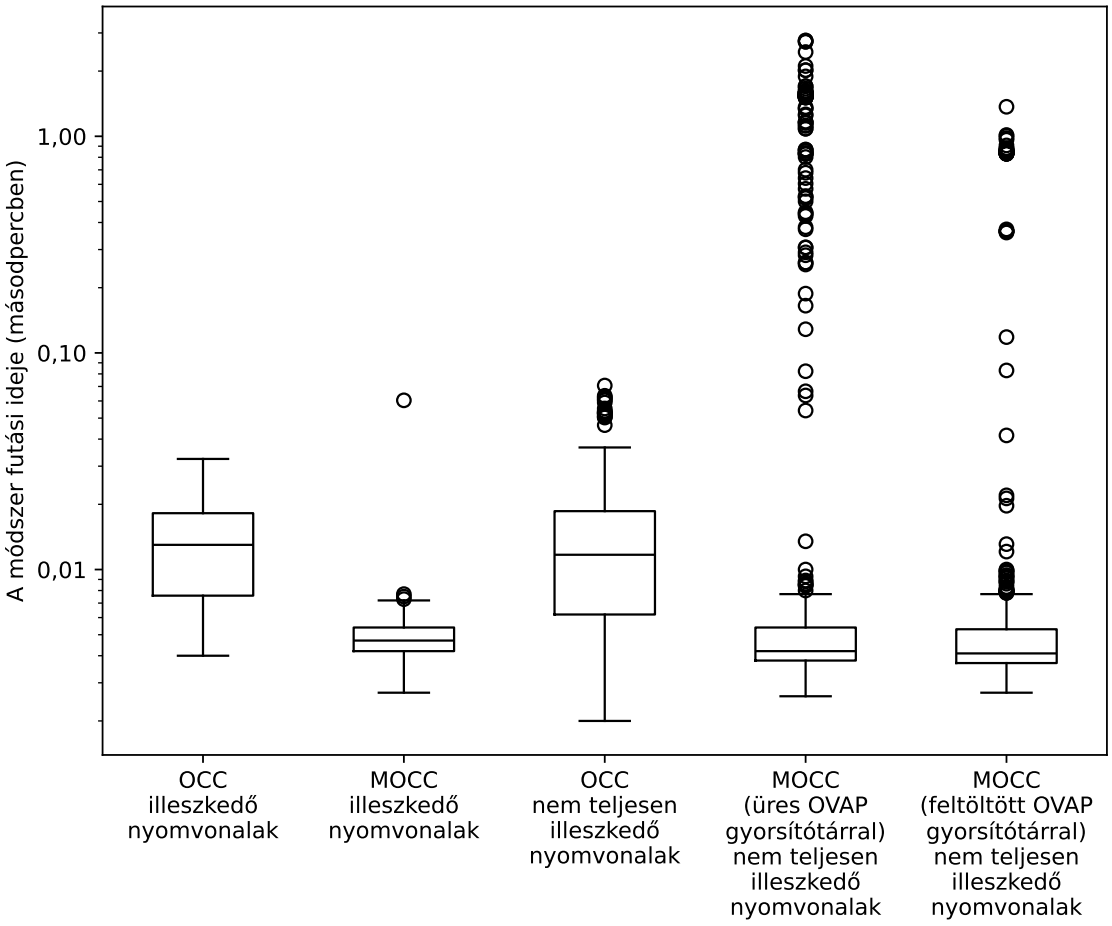
Az OCC vizsgálat eredményei

A tekercs összeszerelési folyamat két eseménynaplójára az OCC és az MOCC módszereket egyaránt 10 alkalommal futtattam le. Az MOCC módszert kezdetben üres, illetve előre feltöltött OVAP gyorsítótárral is teszteltem a hatékonyság összehasonlítása érdekében. A futtatások során mindkét algoritmus sorrendben feldolgozta az összes eseményt, az eseményt hozzáfűzte az ahhoz tartozó eset nyomvonalához, kiszámította az esethez az új optimális előtag-igazítást, majd rögzítette a számítási időt. A másodpercben megadott, 10 lefutásra átlagolt futási időket a 2.9. ábra jeleníti meg dobozdiagram formájában, a használt módszer (OCC vagy MOCC) és a nyomvonal típusa (illeszkedő vagy nem teljesen illeszkedő) szerint csoportosítva. Az átlagolt futási idők statisztikai jellemzőit a 2.2. táblázat foglalja össze. Érdekes megfigyelés, hogy az **illeszkedő nyomvonalak** esetében az MOCC módszer átlagosan gyorsabb volt, mint az OCC módszer. Ennek oka, hogy a javasolt MOCC módszer lehetővé teszi a közvetlen szinkronizálást, így az OCC módszerrel szemben kevesebb állapotot kellett vizsgálnia a keresési folyamat során. Ezzel szemben a **nem teljesen illeszkedő nyomvonalak** esetében, ahol az esemény megfigyelését követően az MOCC módszer nem tudott közvetlen szinkronizálást végrehajtani, a számítási idő hosszabbnak bizonyult. A 2.9. ábrán látható, hogy az üres OVAP gyorsítótárat használó MOCC módszer esetében ezeknek az eseteknek a számítási ideje jól elkülönül a többitől (0,05-nál nagyobb értékek). Az előre feltöltött OVAP gyorsítótárat használó MOCC módszer azonban sikeresen csökkentette a számítási időt ezeknél az eseteknél. Ez annak köszönhető, hogy az OVAP gyorsítótár az ismétlődő OVAP-val rendelkező nyomvonalaknál megspórolta az OVA számítására fordított időt.

Az OCC és az MOCC módszerek által adott online és offline igazítások teljes költségét a nem teljesen illeszkedő nyomvonalak esetén a C. függelék C.5. táblázata foglalja össze. (Az illeszkedő nyomvonalak esetében mindkét módszer egységesen 0 költségű igazításokat adott, ezért ezekről nem készült külön táblázat.) Az online igazítások esetében az OCC módszer az esetek 11%-át, míg az MOCC módszer az

esetek 86%-át jelölte hibásnak. Az offline igazítások esetében az OCC módszer az esetek 28%-át, az MOCC módszer pedig az esetek 100%-át jelölte hibásnak. A táblázatból az is megállapítható, hogy az MOCC módszer által adott igazítások minden nyomvonal esetében magasabb teljes költséggel rendelkeznek, mint az OCC módszer által adott igazítások. A nagyobb teljes költség elsődlegesen arra utalhat, hogy az igazítás nem optimális. Azonban a nyomvonalakat más MCC módszerrel is vizsgáltam (lásd „Az MCC vizsgálat eredményei"), így az optimális igazítási költségek ismertek. Ebben az összefüggésben a magasabb költség azt jelenti, hogy az MOCC módszer – a folyamat többszempontú szemlélete révén – több helytelen viselkedést képes felismerni, mint az OCC módszer.

Az MOCC módszer az 1 másodperces számítási időt csak a nem teljesen illeszkedő nyomvonalak esetében lépte túl, és ezeknél sem minden esetben. Az előre feltöltött OVAP gyorsítótárat alkalmazó MOCC módszer használatakor mindössze három ilyen eset fordult elő, és a számítási idő ezekben az esetekben is elfogadható mértékű maradt (2,5 másodpercnél kevesebb). Feltételezve, hogy valós helyzetekben ritkán fordulnak elő ilyen mértékben nem illeszkedő nyomvonalak, az MOCC módszer aktuális számítási sebessége elfogadhatónak tekinthető.



2.9. ábra. Az OCC és az MOCC módszerek futási ideje különböző illeszkedésű nyomvonalakon (másodpercben megadva)

2.2. táblázat. Az OCC és az MOCC módszerek futási idejének statisztikái a különböző illeszkedésű nyomvonalakra vonatkozóan (másodpercben megadva)

Statisztikai mérőszám	Illeszkedő nyomvonalak		Nem teljesen illeszkedő nyomvonalak		
	OCC	**MOCC**	**OCC**	**MOCC[1]**	**MOCC[2]**
Átlag	0,0136	0,0049	0,0140	0,1706	0,0517
Szórás	0,0074	0,0025	0,0104	0,4750	0,1935
Minimum	0,0040	0,0027	0,0020	0,0026	0,0027
Maximum	0,0324	0,0604	0,0707	2,7673	1,3685

MOCC[1]: üres OVAP gyorsítótárral, MOCC[2]: előre feltöltött OVAP gyorsítótárral.

Az MCC vizsgálat eredményei

Az MCC vizsgálat során a BMCC és az MOCC módszereket mindhárom folyamat folyamatmodelljére és eseménynaplóira lefuttattam. A módszerek által visszaadott több perspektívás igazítások a C. függelék C.2. szakaszában találhatók. Az illeszkedő és a nem teljesen illeszkedő nyomvonalak esetében is egy-egy esetre adott igazítást választottam ki az ábrázoláshoz. Az igazítások színes Chevron-diagramok formájában vannak ábrázolva, ahol a nyílhegyek igazítási mozgásoknak felelnek meg és a színek a mozgások típusát jelölik. A használt színkódolás a következő:

1. zöld: helyes szinkron mozgás,
2. sárga: helytelen szinkron mozgás,
3. piros: eseménynapló mozgás,
4. lila: megfigyelhető tevékenységhez tartozó modell mozgás,
5. szürke: nem megfigyelhető tevékenységhez tartozó modell mozgás (ennek a költsége általában 0).

Az **illeszkedő nyomvonalak** esetében (C.4. ábra, C.5. ábra és C.8. ábra) csak az esetazonosítót tüntettem fel, mivel ezeknél az eseteknél mindkét módszer ugyanazt az igazítást adja eredményül és az igazítások teljes költsége egységesen 0. A **nem teljesen illeszkedő nyomvonalak** esetében (C.6. ábra, C.7. ábra és C.9. ábra) az esetazonosítón kívül a BMCC és az MOCC módszerek által adott igazítások teljes költségét (a tevékenységköltségből és a változóköltségből álló rendezett pár formájában), illetve a két igazítás közötti különbségeket tüntettem fel. Ha mindkét módszer azonos költségű igazítást eredményezett, akkor a költséget az esetazonosító alatt jelenítettem meg. Eltérő költségek esetén a költségeket a megfelelő módszer neve alatt tüntettem fel. A két igazításhoz csak akkor készítettem külön ábrát, ha az igazítási mozgások összetétele és/vagy sorrendje eltért. Az ábrákon kizárólag a hibás változó hozzárendelések szerepelnek. Ezeket az igazítási mozgás alatt jelenítettem meg, ahol a változó neve mellett fekete színnel jelöltem az eseményből származó értéket (ha van), és piros színnel jelöltem a folyamatmodell alapján javasolt értéket. Ha a két módszer által javasolt igazítások között csak a változókhoz rendelt értékek különböztek, akkor a javasolt változó hozzárendeléseket külön sorba írtam.

A közúti közlekedési bírságkezelési folyamat esetében (C.6. ábra) tízből négy nyomvonal esetében eredményezett a két módszer vezérlésfolyam perspektíva szempontjából azonos igazításokat. Ezek közül két esetnél csak a változókhoz hozzárendelt értékek térnek el, míg egy esetnél az érintett változó is különbözik. A többi

nyomvonal esetében a két módszer által javasolt igazítások eltérnek, azonban az igazítások teljes költsége – a tevékenységköltség és a változóköltség összege – megegyezik. Az ábrán látható, hogy az eltérő igazítások esetén a BMCC módszer által adott igazítások tevékenységköltsége nagyobb, mint az MOCC módszeré. Tehát az azonos teljes költségű igazítások közül a BMCC módszer jellemzően azt találja meg, ahol a vezérlésfolyamban van eltérés a nyomvonal és a folyamatmodell között, míg az MOCC módszer azt, ahol a többi perspektívában van az eltérés. További különbség, hogy a változó hozzárendeléseknél a BMCC módszer a lehető legkisebb értéket javasolja, míg az MOCC módszer az eseményből származó attribútumértékhez legközelebb eső elfogadható értéket választja.

A kórházi számlázási folyamat esetében (C.7. ábra) tízből három nyomvonal esetében eredményezett a két módszer azonos igazításokat. A többi nyomvonal esetében a két módszer által javasolt igazítások teljes költsége azonos és (egy nyomvonal kivételével) különbség csak az egymást követő eseménynapló mozgások és modell mozgások sorrendjében van.

Az automatizált tekercs összeszerelési folyamat esetében (C.9. ábra) is a másik két folyamatnál tapasztalt eltérések jelentkeztek, ezért az ábrán csak az MOCC módszer által adott igazításokat tüntettem fel.

Mindegyik folyamatnál a vizsgált nyomvonalak esetében a két módszer azonos teljes költségű igazításokat eredményezett. Az igazítások közötti különbségek nem befolyásolják azok minőségét. Ez alapján megállapítható, hogy a két módszer minőségi szempontból azonos kimenetek előállítására képes.

Limitációk

A javasolt MOCC módszer OCC és MCC megoldásként is elfogadható eredményeket nyújtott, de még vannak fejlesztési lehetőségek. A folyamatváltozók esetében például szükséges lenne az általuk felvehető lehetséges értékek intervallumának lekorlátozása, hogy a módszer által javasolt értékek valósághűbbek legyenek. További nehézséget jelentett a vizsgálatok során az őrkifejezésben szereplő speciális karakterek (pl. „#”) kezelése, amelyet jelenleg csak azok latin betűkre való lecserélésével lehetett megoldani. Emellett a lehetséges optimális teljes költséggel rendelkező igazítások közül a módszer jelenleg csupán egyet képes visszaadni, ami nem feltétlenül tükrözi a valós problémát. Ez ugyan bizonyos mértékben kezelhető a költségfüggvények megfelelő beállításával, de további lehetőséget kínálna a megoldás, ha képes lenne visszajelzéseket fogadni a felhasználótól az igazítással kapcsolatban, például arról, hogy a jelzett problémák valóban relevánsak-e. Ezek figyelembevételével a módszer a jövőben még pontosabb igazításokat biztosíthatna.

2.5.2. A javasolt folyamatadat vizualizációs módszerek hatékonyságának vizsgálata

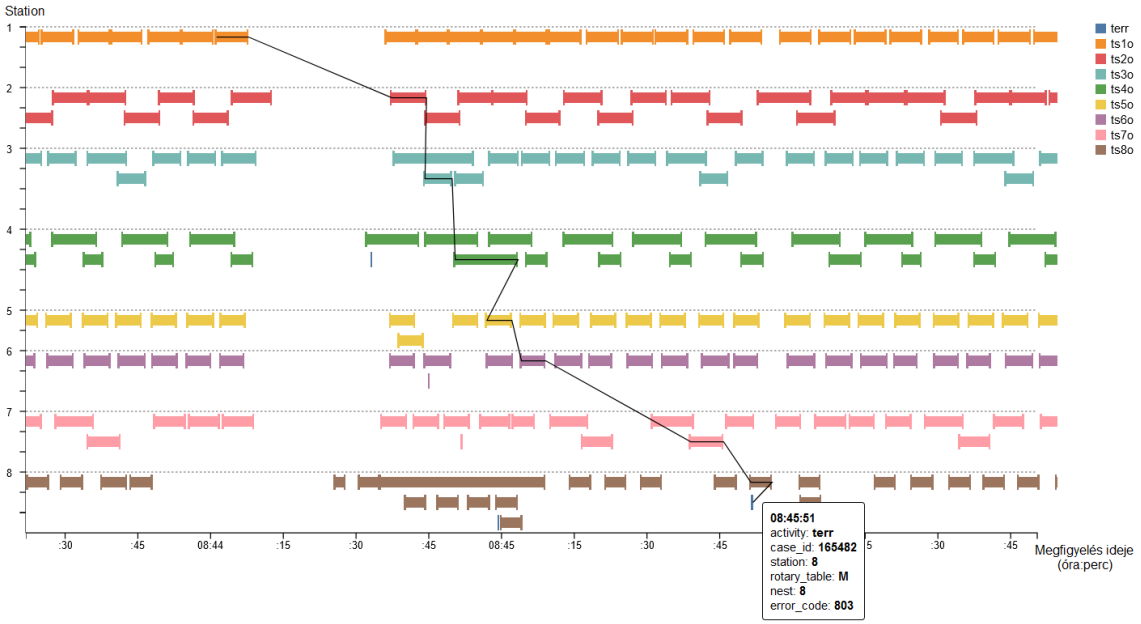
Vizsgálati környezet és paraméterek

Mindkét javasolt folyamatadat vizualizációs módszer teszteléséhez az automatizált tekercs összeszerelési folyamat folyamatmodellje (C. függelék C.1.1. alszakasza) és az eseményadataiból generált naplófájlok lettek felhasználva. Az eredeti eseménynaplók felhasználásával kétféle eseménynapló került létrehozásra:

- Csak a modellre nem teljesen illeszkedő nyomvonalakat tartalmazó eseménynapló: Ez az eseménynapló segít bemutatni, hogy a különböző eltérések az elvárt viselkedéstől hogyan jelennek meg a vizualizációban.
- Egyetlen gyártási nap nyomvonalait tartalmazó eseménynapló: Ez az eseménynapló betekintést nyújt abba, hogy valós életbéli helyzetekben hogyan jelennek meg a vizualizációk.

Eredmények

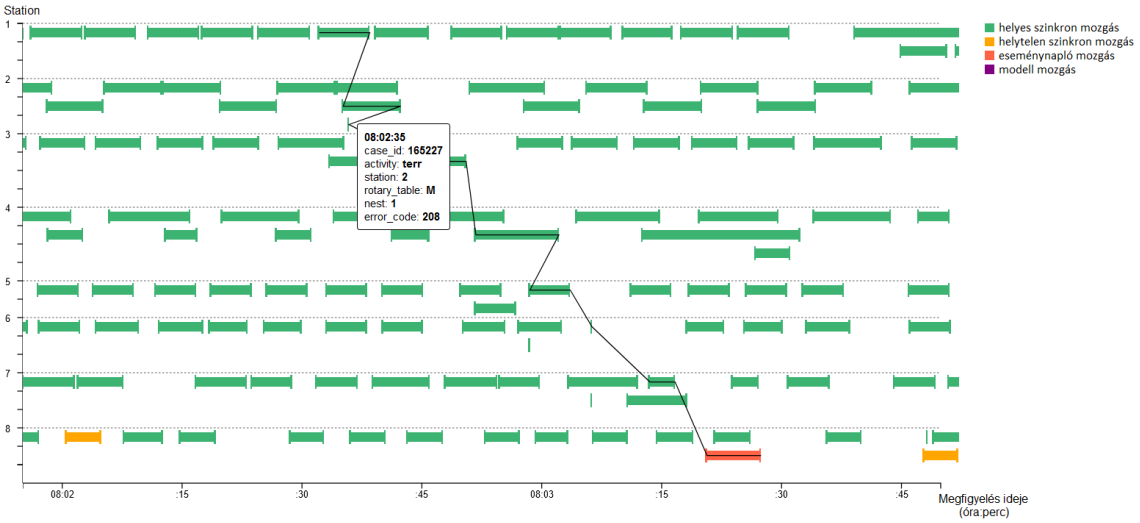
A 2.10. ábrán egyetlen gyártási nap nyomvonalaiból származó eseményadatok vizualizációjának egy része látható, ahol a „station” attribútum van az y-tengelyen megjelenítve. Ennél a folyamatnál a valóságban nem lehetnek átfedő események, mivel a gyártósor minden állomáson egyszerre csak egy termékkel tud foglalkozni. Tehát vagy a műveletek időtartamának mérésével vagy a mért idő rögzítésével van probléma. Néhány állomás esetében, mint például az 5. és a 6. állomás, az időértékek többnyire pontosak, csak időnként van átfedés, valószínűsíthetőleg a gyártósor rövid leállása miatt. Más állomásokon, például a 2–4. állomáson azonban gyakoribbak az átfedések. A 3. állomáson minden hetedik esemény, a 4. állomáson pedig minden második esemény átfedésben van a megelőző eseménnyel. Egy másik időbeli anomália jellemzően a leállások után jelentkezik. A 8. állomáson például egy különösen hosszú esemény figyelhető meg, amely körülbelül 8:44:30-tól 8:45:15-ig tart. Ezzel szemben a 4-7. állomáson szokatlanul rövid események láthatók közvetlenül a kisebb leállást követően.



2.10. ábra. Egyetlen gyártási nap nyomvonalaiból származó eseményadatok vizualizációja

A vizualizációk alapján úgy látszik, hogy a leállások az időbeli anomáliák okozásán kívül a hibás termékek számát is növelik. A 2.10. ábrán egy rövid leállással megszakított folyamatvégrehajtás van kijelölve, ami hibás termékkel zárult. Azonban ez nem az egyedüli ilyen eset. Az ábrán még két további hiba figyelhető meg: az egyik a 4. állomáson 8:44 körül, a másik a 8. állomáson 8:45 körül, ami további két hibás terméket eredményezett.

2.11. ábra. Egyetlen gyártási nap nyomvonalaiból származó igazítási adatok vizualizációja: a leggyakoribb eltérés



2.12. ábra. Egyetlen gyártási nap nyomvonalaiból származó igazítási adatok vizualizációja: a második leggyakoribb eltérés

A 2.11. ábrán és a 2.12. ábrán egyetlen gyártási nap nyomvonalaiból származó igazítási adatok vizualizációjának egy része látható, ahol a „station” attribútum van az y-tengelyen megjelenítve. Látható, hogy a nyomvonalak többsége teljes mértékben illeszkedik a modellre és csak néhány eltérés van az elvárt viselkedéstől. Bár ezek az eltérések ritkán fordulnak elő, folyamatos megjelenésük idővel jelentős problémákhoz vezethet. Megjegyzendő, hogy a legtöbb ilyen eltérés az utolsó állomáson, a 8. állomáson fordul elő. A 2.11. ábrán a leggyakoribb eltéréstípusra látható példa. Az ilyen eltérések mögött a következő két probléma egyike lehet: vagy a késztermék nem kerül a tálcára, vagy a „tray_row” és a „tray_column” értékek nem kerülnek rögzítésre. A folyamatmodell mindkét attribútumhoz 1-es értéket választott, mivel a modellben meghatározott egyetlen kritérium az, hogy ezen értékeknek nagyobbnak kell lennie, mint nulla. A 2.12. ábrán a második leggyakoribb eltéréstípusra látható példa. Egy félkész termék, amely a 2. állomáson hibásnak lett nyílvánítva, nem

került a 7. állomáson eldobásra, hanem eljutott az utolsó állomásig. Mivel mind a „tray_row”, mind a „tray_column” attribútumok értéke 0, ez a 2.11. ábrán is látható állapotot tükrözi, ami arra utal, hogy a termék vagy nem került a tálcára, vagy az értékek nem kerültek rögzítésre.

Amikor a vizualizációs módszereket kizárólag a nem illeszkedő nyomvonalakra alkalmaztam, a megfigyeléseim a korábban leírtakat tükrözték:

- Egy vagy több leállás a folyamat végrehajtások során túl rövid vagy túl hosszú eseményeket eredményezett, ami az érintett termékek hibássá válásához vezetett, vagy legalábbis a rendszer hibásnak nyilvánította azokat.
- A hibásnak nyilvánított félkész termékek a 8. állomásra lettek szállítva ahelyett, hogy a 7. állomáson le lettek volna selejtezve.

A kiváltó okoktól függetlenül kulcsfontosságú a feltárt problémák kezelése annak biztosítása érdekében, hogy minden termék – különösen a jó termékek – kezelése megfelelő módon történjen. Bár az eltérések kiváltó okai nem határozhatók meg egyértelműen kizárólag a vizualizációk segítségével, a lehetséges okok leszűkítésében segítenek. A gyártósor gépkezelői például könnyen ellenőrizhetik a valóságban, hogy a termékek valóban a tálcára kerültek-e vagy sem. Ha igen, akkor a probléma valószínűleg csak adatminőségi probléma. Ha nem, akkor a gép hibásan működik és javításra szorul.

Teljesítményértékelés

A vizualizációs módszerek reakciókészsége elsősorban a megjelenített objektumok (események vagy igazítási lépések) számától függ. Az igazítási lépések száma vagy megegyezik az események számával, vagy meghaladja azt, a modell mozgások teljes száma alapján. A vizsgált folyamatpéldányok esetében modell mozgások ritkán fordultak elő, így a két érték akár egyenlőnek is tekinthető.

Ha bemenetként egy egyszerű eseményfolyamot használunk, az események száma megegyezik a megfigyelt egységek számával. Ezzel szemben egy összetett eseményfolyam esetén az események száma körülbelül kétszerese a megfigyelt egységekéhez képest. Ennek az az oka, hogy egy megfigyelt egység a vizualizáción két külön eseményként jelenik meg, ha a kezdő és a befejező időpontok eltérnek egymástól.

A vizsgált folyamatnál jellemzően csak a hibaesemények (azaz a „t_{err}” események) rendelkeznek egy darab időbélyeggel. Bár olyan időbeli anomáliák alkalmanként előfordulnak, amelyek miatt az állomásműveletek szintén egyetlen időbélyegértékkel rendelkeznek. Azonban – amint az az előző alszakaszban is látható – ezek az esetek ritkák. Ezért most feltételezzük azt, hogy az események száma nagyjából kétszerese a megfigyelt egységek számának.

A vizsgálatok alapján az események optimális mennyisége 1.000 körül van. A vizsgált folyamatok esetében ez körülbelül 20 percnyi eseményadatnak felel meg. Ezzel a határértékkel mind a nagyítás, mind a görgetés zökkenőmentesen működik. Ezenkívül a speciális attribútum, a szűrési attribútum vagy a szűrési attribútum értékének a módosításakor a változások szinte azonnal, jellemzően egy másodpercen belül érvénybe lépnek. Az eszköztipp megjelenítésének a sebességét és az azonos esethez tartozó objektumok összekötésének a sebességét nem befolyásolja az események száma. A módszerek által kezelhető események számának felső határának a meghatározása nagy kihívást jelent. 20 000 esemény beállításánál (ami a vizsgált

folyamatban körülbelül 7 órányi eseményadatnak felel meg) a módszerek még működnek, de a nagyítás és a görgetés észrevehetően lassúvá válik. Ezenkívül a speciális vagy szűrési attribútum módosítása esetén körülbelül egy percet is igénybe vehet, amíg a vizualizációk frissülnek.

Fontos megjegyezni, hogy a vizsgált folyamatnak rövid (ideális esetben 1–2 perc körüli) átfutási ideje van, és 8 párhuzamos folyamatvégrehajtással működik. Ebből adódóan, különböző folyamatokban ugyanaz az eseményszám akár hosszabb időtartamot is felölelhet. Fontos, hogy a bemutatott vizualizációs módszerek elsődleges célja a valós idejű folyamatfelügyelet megkönnyítése. Ennélfogva megfelelőnek tekinthető, ha csak a legutóbbi 20 percet lehet részletesen megvizsgálni.

Limitációk

A javasolt vizualizációs módszerek előnyei közé tartozik, hogy a vizualizált objektumok átfedés nélkül jelennek meg, ami jelentős előrelépést jelent a jelenlegi megoldásokkal szemben. Továbbá a testreszabhatóság lehetőséget biztosít arra, hogy a felhasználók az y-tengelyen különböző attribútumokat jelenítsenek meg, így a folyamat több perspektívából is vizsgálható. A módszer korlátai azonban nagy mennyiségű vizualizált objektum esetén megmutatkoznak, mivel ilyenkor a vizualizáció teljesítménye lassulhat. Emiatt szükséges az egyszerre megjelenített objektumok számának korlátozása. Emellett, bár az eltérések vizuálisan jól követhetők, a rendszer nem nyújt automatikus figyelmeztetést a felhasználó számára.

2.6. Összefoglalás

Ebben a fejezetben az általam kifejlesztett előtag-igazítás alapú több perspektívás online megfelelőség-ellenőrzési (MOCC) módszert, valamint a folyamatadatokat (az eredeti eseményadatokat és az igazítás alapú megfelelőség-ellenőrzések kimenetét) vizualizáló módszereket mutattam be és teszteltem.

A javasolt MOCC módszer

A javasolt MOCC módszer célja, hogy támogassa az eseményadatok valós idejű nyomon követését, különböző nézőpontokból. A megoldás egy (adat Petri-háló formájában megadott) több perspektívás folyamatmodell és egy eseményfolyam megfigyelt eseményei közötti optimális több perspektívás előtag-igazításokat ad vissza. Ehhez az irodalomban felelhető inkrementális A* algoritmust fejlesztettem tovább, ami eredetileg csak a vezérlésfolyam perspektívát veszi figyelembe. A számítási idő csökkentése érdekében az MOCC módszer két funkcióval láttam el: közvetlen szinkronizálás és OVAP megoldások gyorsítótárazása. Az előbbi a teljesen illeszkedő nyomvonalak esetén gyorsítja meg a keresési folyamatot, az utóbbi pedig a nem teljesen illeszkedő nyomvonalak esetén. Online környezetben folyamatban lévő és befejezett folyamatlefutások is lehetnek, amik eltérő bánásmódot igényelnek. Emiatt hoztam létre az eset gyorsítótárazási módszert.

Az MOCC megoldás futási idejét és kimenetinek (azaz a több perspektívás előtag-igazításoknak) minőségét több valós folyamaton teszteltem. Az eredmények azt mutatták, hogy több perspektívás és online megoldásként is megfelelő. A folyamat többszempontú szemlélete miatt képes több helytelen viselkedést felismerni, mint

egy OCC megoldás. A befejezett folyamatlefutásokhoz azonos minőségű több perspektívás igazításokat nyújt, mint egy MCC megoldás, de gyorsabban. Továbbá, mivel képes kezelni a még folyamatban lévő folyamatlefutásokat is, így – egy MCC megoldással ellentétben – azoknak a megfelelőségéről is tud információt nyújtani. Ebből adódóan, (a megfelelő beállításokkal) a javasolt MOCC módszer ideális rövid átfutási idővel és előírásos folyamatmodellel rendelkező valós folyamatok nyomon követésére, de hosszabb átfutási idővel rendelkező folyamatokra is alkalmazható.

A javasolt folyamatadat vizualizációs módszerek

A javasolt folyamatadat vizualizációs módszerek célja, hogy intuitív és átfogó képet nyújtsanak az összetett folyamatadatokról, így lehetővé téve a vizsgált folyamat nyomon követését, valamint az eltérések és azok kiváltó okainak gyors felderítését. A módszerek az eredeti eseményadatok és az igazítási adatok megjelenítésére egy Gantt-diagram által inspirált vegyes (fix és időtartamos) idővonal-alapú vizualizációt használnak, amelyek az átfedő objektumokat is kezelik. Emellett lehetővé teszik a felhasználók számára, hogy kiválaszthassák, milyen perspektívából szeretnék vizsgálni az adatokat.

A javasolt vizualizációs módszereket tesztkörnyezetben valósítottam meg. Hasznosságukat és teljesítményüket egy valós gyártási folyamaton teszteltem. Az eredmények azt mutatták, hogy a módszerek alkalmazásával könnyen kimutathatók az amúgy nehezen fellelhető rendellenségek és a problémák (pl. termék meghibásodás) lehetséges kiváltó okai (pl. váratlan rövid gépleállás a folyamat végrehajtása közben) is.

3. fejezet

Statikus és dinamikus kapacitáskorlátos ív útvonaltervezési probléma

3.1. Irodalmi áttekintés

Ebben a szakaszban a kapcsolódó munkák kerülnek bemutatásra és elemzésre. A 3.1.1. alszakaszban a CARP számára kifejlesztett algoritmusok kerülnek bemutatásra. A 3.1.2. alszakaszban a DCARP számára kidolgozott megközelítések kerülnek összefoglalásra. A 3.1.3. alszakaszban a CARP-hoz hasonló problémákra kifejlesztett ABC algoritmusok kerülnek bemutatásra.

3.1.1. A CARP számára kifejlesztett algoritmusok

Amint azt a bevezetésben már említettem, a CARP-ra főként megközelítő módszerek (azaz heurisztikák és metaheurisztikák) léteznek. Ennélfogva ebben az alszakaszban csak az ebbe a kategóriába tartozó módszerek kerülnek említésre.

Heurisztikák

Golden és szerzőtársai fejlesztették ki az első heurisztikus algoritmusokat a CARP számára, konkrétan az útkereső (*path-scanning*) és a bővítő-egyesítő (*augment-merge*) algoritmusokat [118].

A CARP további nevezetes heurisztikái a párhuzamos beillesztési módszer (*parallel-insert method*) [119], az Ulusoy-féle túrafelosztási módszer (*Ulusoy's tour splitting method*) [120], a bővítési-beillesztési módszer (*augment-insert method*) [121], az útkeresés ellipszis szabállyal [122] és az útkeresés hatékonysági szabállyal [123].

Metaheurisztikák

A CARP metaheurisztikus algoritmusai (néhány kivételtől eltekintve) két fő kategóriába sorolhatók: trajektória-alapú és populáció-alapú.

A trajektória-alapú algoritmusok közül figyelemre méltó az irányított helyi keresési algoritmus [124], a tabu keresési algoritmusok [125, 126], a változó szomszédság keresési algoritmus [127], és a mohó, véletlenszerű adaptív keresési eljárás evolúciós

útvonal-újrakapcsolással [128]. Megemlítendő, hogy a [126] publikációban a tabu kereső algoritmus (*Tabu Search Algorithm*, röviden *TSA*) két változatát mutatták be (TSA1 és TSA2), amelyek közül az utóbbi teljesített jobban. A [129] publikációban egy globális javítási műveletet fejlesztettek ki és építettek be a TSA-ba, létrehozva a javításalapú tabu keresést (*Repair-based Tabu Search*, röviden *RTS*), amely felülmúlja a TSA-t.

A populáció-alapú algoritmusok közül kiemelkedik a genetikus algoritmus [130], a memetikus algoritmusok [131, 33] és a hangya kolónia optimalizációs algoritmusok [132, 133, 37]. Ezek közül a memetikus algoritmus kiterjesztett szomszédsági kereséssel (*Memetic Algorithm with Extended Neighborhood Search*, röviden *MAENS*) [33] a legnépszerűbb, annak ellenére, hogy csak viszonylag jó minőségű megoldásokat ad, és emellett a futási ideje lassú. Több olyan megoldás létezik, amely a MAENS egyes részein próbál javítani (pl. [34, 35]), de ezek a javítások nem igazán növelik a módszer teljes teljesítményét. A hangya kolónia optimalizációs algoritmus útvonal-újrakapcsolással (*Ant Colony Optimization Algorithm with Path Relinking*, röviden *ACOPR*) [37] csak viszonylag jó minőségű megoldásokat ad, de jelenleg ez rendelkezik a leggyorsabb futási idővel.

A hibrid metaheurisztikus megközelítés (*Hybrid Metaheuristic Approach*, röviden *HMA*) [36] egy populáció-alapú algoritmus, amely a helyi finomítási eljárás részeként randomizált tabu-küszöbértékelési eljárást alkalmaz. A HMA az összes létező algoritmus közül a legjobb minőségű megoldásokat adja, és gyorsabb futási idővel rendelkezik, mint a MAENS, de még mindig viszonylag lassú néhány valós életen alapuló CARP példa esetében. Az ACOPR csak viszonylag jó minőségű megoldásokat ad, de jelenleg a leggyorsabb futási idővel rendelkezik. Legalábbis az összehasonlító tesztkészletekből származó CARP példák többsége esetében ez igaz.

3.1.2. Megközelítések a DCARP számára

A DCARP fontossága ellenére viszonylag kevés olyan CARP (vagy ARP) témájú tanulmány van, amely figyelembe veszi a probléma dinamikus változásait a megoldás végrehajtása során [134, 135, 136, 137, 138, 139, 140, 141, 46, 52]. Ráadásul csak három olyan tanulmány van, amely kettőnél több típusú változást vesz figyelembe [138, 46, 52], és közülük csak kettő (beleértve az egyik saját munkámat is) veszi figyelembe az összes kritikus változást, amely bekövetkezhet [46, 52]. (A lehetséges események és azok részletes elemzése, valamint a meglévő D(C)ARP megközelítések összehasonlítása a D. függelék D.1. szakaszában olvasható.) A kritikus változások vagy események olyan mértékben megváltoztathatják a problémát, hogy a jelenlegi megoldás már nem kivitelezhető, ezért ezek kezelése elengedhetetlen. Mind [46], mind [52] olyan keretrendszert javasol a DCARP számára, amely bonyolult speciális algoritmusok helyett lehetővé teszi bármely statikus CARP megoldó használatát egy DCARP példa megoldására. Legjobb tudomásom szerint az [52]-ben bemutatott adat-vezérelt DCARP megoldás az egyetlen adat-vezérelt megoldás a DCARP vagy akár a CARP számára.

3.1.3. Az ABC algoritmus és alkalmazásai

Az eredeti ABC algoritmust [38] Karaboga vezette be. A [142] publikációban Karaboga és Görkemli új definíciót javasolt a megfigyelő méhek keresési viselkedésére,

ami javította az algoritmus konvergencia-teljesítményét. Ezért az ABC algoritmus új változatát gyors ABC-nek (*quick Artificial Bee Colony*, röviden *qABC*) nevezték el.

Az ABC algoritmust többváltozós és többmódusú folytonos függvények optimalizálására alkalmas algoritmusként vezették be, de később sikeresen alkalmazták más típusú optimalizálási problémákra is. Karaboga és Görkemli bevezetett egy ABC és egy qABC algoritmust kombinatorikus problémákra (CABC, illetve qCABC), amiket az utazó ügynök problémára (*Traveling Salesman Problem*, röviden *TSP*) alkalmaztak [39, 40]. Mindkét algoritmus a mohó résztúra mutációs (*Greedy Sub Tour Mutation*, röviden *GSTM*) műveletet használja [143], amelyet a TSP-t megoldó genetikus algoritmus teljesítményének növelésére fejlesztettek ki. Bebizonyították, hogy a GSTM jelentősen gyorsabb és pontosabb, mint más létező mutációs műveletek [143]. Továbbá kimutatták, hogy az ABC és a qABC algoritmusok jobb teljesítményt nyújtanak, mint számos evolúciós számításon alapuló optimalizációs algoritmus [42]. Mivel a TSP hasonló a CARP-hoz, abban a reményben, hogy egy GSTM-hez hasonló művelettel rendelkező ABC algoritmus jól fog teljesíteni, kifejlesztettem a CARP-ABC algoritmust (3.4. szakasz) a részútvonalterv művelettel (3.3. szakasz).

A jármű útválasztási problémához (*Vehicle Routing Problem*, röviden *VRP*) is léteznek ABC algoritmusok (pl. [144]) és a VRP különböző változataihoz is (pl. [43, 44]). A CARP-ra azonban csak egy ABC algoritmus létezik, és az is csak a CARP egy speciális változatára, a nem irányított nyereséggel rendelkező CARP-ra [41]. Legjobb tudomásom szerint tehát jelenleg sem a CARP-ra, sem a DCARP-ra nem létezik ABC algoritmus.

3.2. Előzetes fogalmak

Ez a rész a javasolt CARP-ABC algoritmushoz kapcsolódó fogalmakat mutatja be, hogy segítsen megérteni annak működését. A fogalmak itt csak röviden kerülnek bemutatásra, a részletesebb leírások a hivatkozott munkákban megtalálhatók.

Ebben a szakaszban először a statikus CARP (3.2.1. alszakasz), majd az (adat-vezérelt) DCARP (3.2.2. alszakasz) kerül megfogalmazásra. Ezt követően az általános ABC-algoritmus (3.2.3. alszakasz), majd a javasolt CARP-ABC algoritmusban is használt, az irodalomban már jólismert mozgási műveletek (3.2.4. alszakasz) bemutatására kerül sor. Legvégül a DCARP során használt esemény és eseményfolyam (3.2.5. alszakasz) kerül definiálásra. A használt jelölések a „Jelölések" között megtalálhatók.

Ebben a szakaszban saját hozzájárulásaim a következők:

- A 3.2.2. alszakaszban az adat-vezérelt DCARP fogalmának bevezetése, amely a [46] publikációban bemutatott DCARP fogalmán alapul.
- A 3.2.5. alszakaszban bemutatott esemény és eseményfolyam fogalmának bevezetése.

3.2.1. A CARP

A meglévő munkákban a bemeneti gráfként egyesek irányítatlan gráfot [36], mások irányított gráfot [145, 37], megint mások pedig vegyes gráfot [130, 33] feltételeznek.

Ebben a munkában irányított gráfot feltételezünk, amelyben az irányítatlan élek két ellentétesen irányított élnek tekintendők.

A CARP (irányított) gráfja a következőképpen írható le: $G = (V, A)$, a csúcsok V halmazával és az ívek (irányított élek) A halmazával. Adott egy $T \subseteq A$ feladathalmaz is, amely meghatározza azokat az íveket, amelyekhez feladatokat rendeltünk. Ha egy CARP példa gráfja (irányítatlan) éleket tartalmaz, akkor az élek ívpárok (azaz élenként és irányonként egy-egy ív) formájában kerülnek hozzáadásra az A halmazhoz. Például, ha (v_i, v_j) egy él és $v_i, v_j \in V$, akkor a (v_i, v_j) és (v_j, v_i) ívek kerülnek hozzáadásra az A halmazhoz. Hasonlóképpen, ha (v_i, v_j) egy él, amelyhez feladatok vannak rendelve, és $v_i, v_j \in V$, akkor a (v_i, v_j) és (v_j, v_i) ívek kerülnek hozzáadásra a T halmazhoz. A gráfnak van egy speciális v_0 csúcspontja is ($v_0 \in V$), a telephely, valamint egy $t_0 = (v_0, v_0)$ álfeladata is, ami egy hurokél. A t_0 álfeladat egy olyan feladat, ami nem valós végrehajtandó feladatot jelöl; a megoldás reprezentációjában használjuk a költségszámítás megkönnyítése érdekében.

A feladatokat w homogén q kapacitású járművekből álló flotta végzi el. Minden jármű útvonala a telephelyről (v_0) indul és ott is fejeződik be. Minden feladatot egyetlen műveletben kell végrehajtani, és minden jármű legfeljebb annyi igényt tud kielégíteni, amennyi a maximális kapacitása.

A gráf megfeleltethető egy úthálózatnak, ahol az ívek útszegmensek. Az útszegmensek egy részéhez feladatok vannak rendelve. A feladatok teljesítéséhez különböző mennyiségű, de azonos típusú igényt kell kielégíteni a járműveknek. Minden ívet a következő függvények jellemeznek:

- *head*: az ív fejcsúcsa;
- *tail*: az ív végcsúcsa;
- *dc*: az áthaladási költség, az adott íven való áthaladás költsége.

Ezenkívül minden feladatot a következő funkciók jellemeznek:

- *id*: a feladat egyedi azonosítója, amely egy pozitív egész szám;
- *dem*: a (pozitív) igény, amely a feladat kiszolgálásához szükséges igénybevételt jelöli;
- *sc*: a szolgáltatási költség, amely az ívhez rendelt feladatok végrehajtásának és az ív áthaladásának a költsége (azaz az *sc* magában foglalja a *dc*-t).

Bár egy él két ellentétes irányú ívnek tekinthető, ha egy feladat van hozzárendelve, akkor a feladatot csak egyszer kell végrehajtani, bármelyik irányban. Legyen $t \in T$ egy él egyik ívének feladata, akkor $inv(t)$ jelölje t inverzét, az él másik feladatát. Ha $head(t)$ és $tail(t)$ a t ívfeladat fej- és végcsúcsa, akkor $head(inv(t)) = tail(t)$ és $tail(inv(t)) = head(t)$ az $inv(t)$ fej- és végcsúcsa. A *dc*, *dem* és *sc* értékek azonosak t és $inv(t)$ ívfeladatok esetében.

Legyen a legalább egy jármű által végrehajtandó feladatok teljes száma n. Az n értéke függ a T halmaz összetételétől: ha a T halmaz csak élekből származó ívfeladatokat tartalmaz, akkor $n = |T|/2$, ha viszont csak ívekből származó ívfeladatokat tartalmaz, akkor $n = |T|$.

A két csúcs közötti minimális teljes áthaladási költséget az $mdc : V \times V \to N$ függvény határozza meg, amely a Dijkstra algoritmust használja a legrövidebb útvonal megkereséséhez. Például, $mdc(v_i, v_j)$ a v_i csúcsból a v_j csúcsba vezető minimális áthaladási költséget jelöli, ahol $v_i, v_j \in V$.

Egy CARP példa (I) a következőképpen definiálható:

$$I = (V, v_0, A, T, n, w, q, head, tail, dc, id, dem, sc, inv, mdc) \tag{3.1}$$

Megoldás reprezentáció

Egy CARP példa megoldása útvonaltervek halmazaként fejezhető ki. Az útvonaltervek az adott sorrendben végrehajtandó $t \in T$ (ív)feladatok sorozatai. Az egymást követő feladatokat az *mdc* függvény által biztosított legrövidebb útvonalak kötik össze. Ennélfogva, egy CARP példa S megoldása a következőképpen írható le:

$$S = \{r_1, r_2, \ldots, r_{|S|}\} \tag{3.2}$$

ahol $|S|$ az útvonaltervek száma és r_k ($k \in \{1, 2, \ldots, |S|\}$) a k-adik útvonalterv az S megoldáson belül. A k-adik útvonalterv a következőképpen írható le:

$$r_k = \langle t_0, t_{k,1}, t_{k,2}, \ldots, t_{k,l_k}, t_0 \rangle \tag{3.3}$$

ahol l_k a (nem ál)feladatok száma és $t_{k,i}$ az i-edik feladat a k-adik útvonaltervben. Megjegyzendő, hogy itt k egy index, amely csak egy adott útvonalterv azonosítására szolgál a megoldásban. Az útvonaltervek sorrendje a megoldáson belül nincs hatással a megoldás minőségére.

Minden jármű útvonalának a telephelyen kell kezdődnie és végződnie, ezért a t_0 álfeladatnak – amely azt jelöli, hogy a jármű éppen a telephelyen tartózkodik – az útvonaltervekben mindig az első és az utolsó elemként kell szerepelnie. Mivel a t_0 álfeladat egy állapotot fejez ki és nem valós feladat, ezért az ív fej- és végcsúcsa azonos, valamint a hozzátartozó költség értékek (*dc*) és (*sc*) és az igény érték (*dem*) is 0-ák. Azonosítójaként (*id*) is a 0 használatos.

Egy CARP példa megoldásának reprezentációjához (a legtöbb járműútvonaltervezési problémához hasonlóan) természetes kódolási módszer használatos. Ez azt jelenti, hogy minden útvonaltervet a feladatok azonosítóinak rendezett listájaként lehet kódolni, így a megoldás e listák láncolásaként reprezentálható. Azonban minden útvonalterv a t_0 álfeladattal kezdődik és végződik, így ha a kódolt útvonalterveket egymáshoz kapcsoljuk, akkor a kapott listában egymást követő álfeladatok vannak. Az egyszerűség kedvéért, a kódolt megoldásban az egymás után következő álfeladatokból csak egy kerül megtartásra.

Az alábbi ábrán (3.1. ábra) egy egyszerű példa található a megoldás reprezentációjára. A példában egy 10 feladatot tartalmazó CARP példa megoldásának reprezentációja látható, ahol 0 az álfeladat azonosítója. Ebben a megoldásban 3 útvonal van. Az 1. útvonal az 1, 4 és 10 azonosítójú feladatokat, a 2. a 8, 2, 7 és 3 azonosítójú feladatokat, a 3. pedig a 6, 9 és 5 azonosítójú feladatokat végzi el.

0	1	4	10	0	8	2	7	3	0	6	9	5	0

3.1. ábra. Példa egy CARP példa megoldásának reprezentációjára

Célkitűzés és korlátozások

A CARP célja a megoldás teljes költségének minimalizálása néhány, ebben az alszakaszban meghatározott korlátozás mellett. Az S megoldás teljes költsége (azaz $TC(S)$) a következő képlettel számítható ki:

$$TC(S) = \sum_{k=1}^{|S|} DC(r_k) + SC(r_k) \tag{3.4}$$

$$DC(r_k) = mdc\Big(t_0, head(t_{k,1})\Big) + \sum_{i=1}^{l_k-1} mdc\Big(tail(t_{k,i}), head(t_{k,i+1})\Big) + mdc\Big(tail(t_{k,l_k}), t_0\Big) \tag{3.5}$$

$$SC(r_k) = \sum_{i=1}^{l_k} sc(t_{k,i}) \tag{3.6}$$

ahol $DC(r_k)$ és $SC(r_k)$ az r_k útvonalterv teljes áthaladási és szolgáltatási költsége.

Az S megoldásnak a következő korlátozásoknak kell megfelelnie. Egyrészt, minden útvonalterv a telephelyen kezdődik és ott is végződik. Másrészt, minden $t \in T$ feladat pontosan egyszer kerül végrehajtásra. Ebből adódóan, az egyes útvonaltervekben szereplő feladatok (kivéve a t_0 álfeladatot) teljes számának egyenlőnek kell lennie n-nel:

$$\sum_{k=1}^{|S|} l_k = n \tag{3.7}$$

Ezenfelül, egy feladat nem szerepelhet többször, sem ugyanazon az útvonalon, sem egy másik útvonalon:

$$t_{a,i} \neq t_{b,j}, \forall (a,i) \neq (b,j) \tag{3.8}$$

ahol r_a és r_b az S megoldáson belüli útvonaltervek, $t_{a,i}$ az r_a útvonalterv i-edik feladata, $t_{b,j}$ pedig az r_b útvonalterv j-edik feladata. Ha egy $t \in T$ feladatnak van inverze (azaz $\exists inv(t)$), akkor vagy t vagy $inv(t)$ szerepel a megoldásban, mindkettő nem szerepelhet ugyanabban a megoldásban. Harmadrészt, az egyes útvonaltervek által kielégítendő teljes igény nem haladja meg a hozzárendelt jármű teljes q kapacitását:

$$\sum_{i=1}^{l_k} dem(t_{k,i}) \leq q, \qquad \forall k \in \{1, 2, \ldots, |S|\} \tag{3.9}$$

3.2.2. Az adat-vezérelt DCARP

A DCARP-nak különböző megközelítései vannak, de ebben a munkában az általam kifejlesztett adat-vezérelt DCARP-ot [52] használom.

Ebben a problémában egy statikus CARP példa helyett egy sorozatnyi DCARP példa (azaz egy DCARP forgatókönyv [46]) van, amelyet meg kell oldani. Egy DCARP forgatókönyvet a $\mathcal{I} = \langle I_0, I_1, \ldots, I_i, \ldots, I_{m-1} \rangle$ jelöl, ahol m a DCARP példák száma a forgatókönyvben (azaz a problémát módosító dinamikus események száma $m-1$). Minden egyes I_i ($I_i \in \mathcal{I}$) DCARP példa az aktuális problémára vonatkozó összes információt tartalmazza. Az I_i DCARP példát az előző I_{i-1} DCARP példa, annak az elfogadott megoldásának az addigi végrehajtása, illetve az elfogadást követően bekövetkezett esemény(ek) határozzák meg, ahol $0 < i < m$. A kezdeti példa (I_0) statikus (adat-vezérelt) CARP példának tekinthető, mivel kezdetben minden jármű a telephelyen van (jó állapotban), és még egy feladatot sem hajtottak végre.

Egy adat-vezérelt DCARP példa esetében az összes járműről és útvonaltervről kell információt tárolni. Minden jármű esetében ismerni kell annak aktuális helyét és állapotát. Továbbá, a járművekhez és az útvonaltervekhez azonosítókat kell használni, mivel egy jármű több útvonaltervet is követhet (egymás után), és minden egyes útvonalterv esetében fontos tudni, hogy az milyen végrehajtási állapotban van (egy jármű már végrehajtotta, a végrehajtása még folyamatban van, vagy a végrehajtás megkezdéséhez még hozzá kell rendelni egy járművet).

A járművek száma (w) helyett az összes jármű azonosítóinak halmazát kell meghatározni, amit a H jelöl. A (jelenleg) szabad járművek azonosítóinak halmazát a H_f ($H_f \subseteq H$) jelöli, amely kezdetben azonos a H halmazzal. Egy jármű azonosítója akkor adódik hozzá a H_f halmazhoz, ha az befejezte a hozzárendelt útvonalterv végrehajtását, és akkor kerül eltávolításra onnan, ha a járműhöz új útvonalterv kerül hozzárendelésre. A (jelenleg) hibás járművek azonosítóinak halmazát a H_e ($H_e \subseteq H$) jelöli, amely kezdetben egy üres halmaz (azaz $H_e = \emptyset$). Ha az összes feladat végre lett hajtva, és minden jármű visszatért a telephelyre (azaz nincsen egyetlenegy lerobbant jármű sem az utakon), akkor $H_f = H$, ellenkező esetben $H_f \cup H_e = H$.

Az összes útvonalterv azonosítóinak halmazát az R jelöli. A (jelenleg) nem módosítható és végrehajtás alatt nem álló útvonaltervek azonosítóinak halmazát az R_e jelöli ($R_e \subseteq R$). Amikor egy új útvonalterv létrehozásra kerül, annak az azonosítója az R halmazhoz adódik hozzá. Amikor egy terv végrehajtása befejeződik vagy felfüggesztésre kerül (jármű meghibásodása miatt), annak az azonosítója az R_e halmazhoz is hozzáadódik. Ha az összes útvonalterv végrehajtása befejeződött, és nincs több végrehajtandó feladat, akkor $R_e = R$, ellenkező esetben $R_e \subset R$. Az azonosító csak akkor kerül ki az R_e halmazból, ha a hozzárendelt jármű meghibásodott, de megjavították, és folytathatja a terv végrehajtását. A függvény, amely megadja, hogy melyik jármű lett hozzárendelve az adott útvonaltervhez, a következő képpen van definiálva: $rv : R \to H$.

A járművek aktuális helyének példában történő tárolására a [46] publikációban bevezetett virtuális feladat stratégiát használtam, amely az egyes útvonaltervekben az azóta végrehajtott feladatokat úgynevezett „virtuális feladatokkal" helyettesíti. A „virtuális feladat" egy olyan ív, amelynek a fejcsúcsa a v_0 telephelycsúcs, a végcsúcsa pedig a jármű aktuális helye, a v ($v \in V$) csúcs. Az egyszerűség kedvéért azt feltételezzük, hogy bármely váratlan esemény bekövetkezésekor minden jármű pont egy csúcson található. Mivel ez a feladat „virtuális", nem lehet áthaladni rajta, ezért végtelen áthaladási költséggel rendelkezik (azaz $dc(v) = \infty$). Továbbá, mivel ez egy „feladat", van hozzárendelve egy kereslet és egy kiszolgálási költség, amelyeket a megadott adatok alapján számítunk ki. A kiszolgálási költség a jármű által eddig termelt összes költség (azaz a jármű által áthaladt vagy kiszolgált ívek áthaladási és kiszolgálási költségeinek összege), az igény pedig a jármű által eddig kielégített összes igény. Egy útvonalterv legfeljebb egy virtuális feladatot tartalmazhat, ezért ha egy útvonaltervnek már van virtuális feladata, akkor azt a megfelelő jármű által azóta áthaladt ívek és végrehajtott feladatok figyelembevételével frissítjük. Az összes virtuális feladat halmazát T_v ($T_v \subseteq T$) jelöli, és a függvényt, amely meghatározza, hogy melyik virtuális feladat tartozik egy adott útvonaltervhez, $rt : R \to T_v$ jelöli.

A végrehajtandó ívfeladatok halmazát T jelöli. Ha az összegyűjtött információk szerint egy t ($t \in T$) feladatot egy h ($h \in H$) jármű végrehajtott, akkor az új DCARP példában a t feladatot el kell távolítani a T halmazból. Továbbá, a jármű útvonaltervének virtuális feladatát (pl. $t_{k,v}$, ahol $rv(k) = h$ és $t_{k,v} \in T_v$) frissíteni kell. Az új virtuális feladatot úgy generáljuk, hogy a t feladat szerepeljen benne (a jármű által végrehajtott egyéb feladatokkal és a jármű által bejárt ívekkel együtt). Ha a t feladatnak van inverze (azaz $\exists inv(t)$), akkor azt is eltávolítjuk a T halmazból. Ennek megfelelően a végrehajtandó feladatok teljes száma (n) eggyel csökken. Hasonlóan, egy t ($t \in T$) feladat lemondása esetén a t feladatot el kell távolítani a T halmazból és az érintett jármű útvonaltervéből, így a végrehajtandó feladatok teljes száma (n) eggyel csökken. Ekkor az érintett jármű útvonaltervének a virtuális feladatát nem

kell frissíteni.

A kezdeti I_0 DCARP példa hasonló a statikus CARP példához. Az R halmazt és az rv függvényt csak az I_0 példa megoldásának megtalálása után hozzuk létre. A H_f halmaz kezdetben egyenlő H halmazzal, majd az rv alapján az összes olyan járműazonosítót, amely egy útvonaltervhez van rendelve, eltávolítjuk a H_f halmazból. Ebben a szakaszban a H_e, az R_e és a T_v halmazok üres halmazok, ezért az rt függvény is üres függvény. Ezek szerint a kezdeti I_0 DCARP példát a következőképpen definiáljuk:

$$I_0 = (V, v_0, A, T, n, w, q, H, head, tail, dc, inv, dem, sc, mdc) \tag{3.10}$$

A következő I_i DCARP példákat (ahol $0 < i < m$) a következőképpen határozzuk meg:

$$I_i = (V, v_0, A, T, T_v, n, w, q, H, H_e, H_f, R, R_e, rt, rv, head, tail, dc, inv, dem, sc, mdc) \tag{3.11}$$

Egy forgatókönyv felépítése

Egy új DCARP példát akkor hozunk létre és adunk hozzá a DCARP forgatókönyvhöz, amikor egy olyan váratlan esemény történik, amely olyan mértékben megváltoztatja az aktuális problémát, hogy az hatással van az aktuálisan végrehajtott megoldásra. A D. függelék D.1. szakaszában az összes lehetséges eseményt (reális feltételezések alapján) összegyűjtöttem és hatásuk alapján elemeztem.

Feltételezzük, hogy az úthálózat, a járművek száma és a járművek maximális kapacitása nem változhat (legalábbis a megoldás végrehajtása során). Ezért V, v_0, A, $head$, $tail$, inv, q és H egy DCARP forgatókönyv minden DCARP példájában azonos. Feltételezzük, hogy egy DCARP példa megoldásának végrehajtása során két fajta esemény következhet be: várt és váratlan esemény. Egy várt esemény az olyan esemény, amely a szolgáltatási terv szerint kerül végrehajtásra, azaz

- egy feladat elvégzésre kerül (ami megváltoztatja a T-t),
- egy jármű elmozdul (ami megváltoztatja a T_v-t, így az rt-t is), vagy
- egy jármű visszatér a telephelyre (ami megváltoztatja a H_f-t és bizonyos esetekben az rv-t is).

Váratlan események a következők lehetnek:

- egy útszakaszt lezárnak/megnyitnak (ami megváltoztatja a dc-t és az mdc-t),
- a forgalom csökken/növekedik egy adott útszakaszon (ami megváltoztatja a dc-t és bizonyos esetekben az sc-t és az mdc-t is),
- egy feladat törlődik/megjelenik (ami megváltoztatja a T, n, dem és sc-t),
- egy jármű meghibásodik (leáll)/újraindul (ami megváltoztatja az R_e-t),

Az érintett komponensek csak akkor frissülnek, ha új DCARP példát kell előállítani. Ha útvonal újratervezésre kerül sor, akkor R és rv változhat, de a változások csak a következő DCARP példában jelennek meg. Ebből adódóan, egy DCARP forgatókönyv DCARP példái között a T, T_v, n, H_f, R, R_e, rt, rv, dem, sc, dc és mdc különböző lehet.

Mivel váratlan események bekövetkezése miatt a DCARP példa egyes összetevői megváltozhatnak, az optimális megoldás is változhat. Ha a jelenlegi megoldás még mindig megvalósítható, de fennáll a lehetősége annak, hogy van annál jobb megoldás is, a szolgáltatási terv újratervezése opcionális. Viszont, ha a jelenlegi megoldás már nem megvalósítható, akkor mindenképpen újra kell tervezni a szolgáltatási tervet.

Megoldás reprezentáció

A DCARP példák esetében a megoldás reprezentációja nagyrészt ugyanaz, mint a statikus CARP példák esetében. Az egyetlen különbség az, hogy ha egy útvonaltervhez virtuális feladatot rendelünk, akkor a virtuális feladat a második feladat az útvonaltervben (mivel az első feladat mindig a t_0 álfeladat). Például, ha az $r_k = \langle t_0, t_{k,1}, t_{k,2}, \ldots, t_{k,l_k}, t_0 \rangle$ útvonaltervnek k ($k \in R$) az azonosítója, és van egy hozzá rendelt t_{v_k} ($rt(k) = t_{v_k} \in T_v$) virtuális feladat, akkor $t_{k,1} = t_{v_k}$.

Célkitűzés és korlátozások

Minden DCARP példa esetében a célkitűzés és a megkötések nagyrészt ugyanazok, mint a statikus CARP példák esetében. Az egyetlen különbség a második korlátozásnál van, amely megköveteli, hogy az S megoldásban szereplő feladatok teljes száma (a t_0 álfeladat nélkül) egyenlő legyen a még végrehajtandó feladatok számának (n) és a virtuális feladatok teljes számának ($|T_v|$) összegével:

$$\sum_{k=1}^{|S|} l_k = n + |T_v| \tag{3.12}$$

A virtuális feladat attribútumai garantálják, hogy a megoldást kereső algoritmus minden virtuális feladatot mindig közvetlenül egy álfeladat után (tehát az útvonalterv elejére) fog elhelyezni egy (közel optimális) szolgáltatási tervben, így nincs szükség erre vonatkozó korlátozás hozzáadására.

Megoldás keresése

Az adat-vezérelt DCARP keretrendszer lehetővé teszi a szolgáltatási terv újratervezését, ha egy kritikus esemény (azaz olyan váratlan esemény, amely megváltoztathatja az aktuális megoldás megvalósíthatóságát) következik be. Ilyen esemény a feladat megjelenése, az igény megnövekedése és a jármű meghibásodása.

Az adat-vezérelt DCARP keretrendszer lehetővé teszi a statikus CARP példákhoz kifejlesztett megoldás kereső algoritmusok használatát azáltal, hogy az aktuális adat-vezérelt DCARP példát statikus CARP példává alakítja át. Miután a CARP megoldó talált egy (kellően jó) megoldást, a keretrendszer azt adat-vezérelt DCARP megoldássá alakítja át.

Egy adat-vezérelt DCARP példa statikus CARP példává alakítása a következőképpen történik: a jármű- és útvonalterv-azonosítók (azaz H, H_f, R és R_e), valamint a kapcsolódó függvények (rt és rv) elhagyásra kerülnek. Továbbá, a T-ből eltávolítjuk a befejezett és felfüggesztett útvonaltervekhez kapcsolódó virtuális feladatokat. Például, ha t_{v_k} ($rt(k) = t_{v_k} \in T_v$) a k ($k \in R$) azonosítóval rendelkező útvonalterv virtuális feladata és az útvonalterv végrehajtása leállt (azaz $k \in R_e$), akkor t_{v_k}-t eltávolítjuk T-ből (azaz $T \setminus \{t_{v_k}\}$).

A statikus CARP példa megoldás adat-vezérelt DCARP példa megoldássá alakítása a következőképpen működik: a befejezett és felfüggesztett útvonaltervekhez kapcsolódó virtuális feladatok külön útvonaltervekben hozzáadódnak a megoldáshoz, hogy a DCARP forgatókönyv teljes költségét nyomon lehessen követni. Továbbá, ha a megoldáson belül új útvonaltervek jelennek meg, a keretrendszer azonosítókat ad

nekik, és megkísérli mindegyiket egy-egy szabad járműhöz rendelni. A többi útvonalterv esetében a bennük lévő virtuális feladat alapján könnyen megállapítható, hogy melyik útvonalterv azonosítója melyik útvonaltervhez tartozik.

3.2.3. Az alap ABC algoritmus

Ez az alszakasz a [42] publikáció alapján mutatja be a kombinatorikus problémákra vonatkozó alap ABC algoritmust. A [38] publikációban bemutatott eredeti ABC algoritmushoz hasonlóan a mesterséges méheket három csoportba soroljuk:

- dolgozó méhek (*employed bees*), akik a táplálékforrásokat hasznosítják;
- megfigyelő méhek (*onlooker bees*), akik döntést hoznak arról, hogy melyik táplálékforrást válasszák;
- felderítő méhek (*scout bees*), akik véletlenszerűen választanak új táplálékforrást.

Az ABC algoritmusban a táplálékforrás egy megoldásnak felel meg, a táplálékforrás nektármennyisége pedig a megoldás megfelelőségének. Az ABC algoritmus egy iteratív eljárás, amely összesen négy fázisból áll. Az inicializálási fázissal kezdődik, majd három méhfázis ismétlődik (mindig ugyanabban a sorrendben), amíg egy előre meghatározott befejezési kritérium nem teljesül. Az inicializálási fázisban a populációt véletlenszerűen generált táplálékforrásokkal inicializálja. Az első fázisban, a dolgozó méhek fázisában, a dolgozó méheket a táplálékforrásokhoz küldi, ahol azok meghatározzák a táplálékforrások nektármennyiségét. A második fázisban, a figyelő méh fázisban, a források valószínűségi értékét a nektármennyiségük alapján kiszámítja, majd a figyelő méheket a legkedvezőbb táplálékforráshoz küldi, hogy megkeressék a szomszédos táplálékforrásokat, és meghatározzák azok nektármennyiségét. A harmadik fázisban, a felderítő méhek fázisában, a méhek által kimerített források kiaknázási folyamata leáll, és a felderítő méheket kiküldi, hogy véletlenszerűen új táplálékforrásokat fedezzenek fel a keresési területen belül. Minden egyes fázisban az eddig talált legjobb táplálékforrást megjegyzi. A fázisok részletesebb leírása a D. függelék D.2. szakaszában található.

3.2.4. Mozgási műveletek a CARP-hoz

A populáció alapú evolúciós algoritmusokban a populáció változatosságának növelése érdekében különböző lépésméretű mozgási műveleteket (*move operators*) alkalmazunk az új, egymáshoz közeli megoldások létrehozására. Ezek a mozgási műveletek két fő kategóriába sorolhatók: kis lépésméretű és nagy lépésméretű műveletek. A kis lépésméretű műveletek egy vagy két útvonalterven belül módosíthatják a feladatok elhelyezkedését és/vagy irányát. Ezzel szemben a nagy lépésméretű műveletek több mint két útvonaltervet képesek módosítani. A szakirodalomban leggyakrabban használt kis lépésméretű műveletek, amelyeket ebben a munkában is használok, az inverzió (*inversion*), az (egyszeri) betoldási (*insertion*), a csere (*swap*) és a kétopciós (*2-opt*) műveletek [33, 36, 37]. Az egyetlen nagy lépésméretű művelet, amelyet ebben a munkában használok, az egyesítés-szétválasztás (*merge-split*) művelet, amelyet [33] publikációban mutattak be. Azért nevezzük nagy lépésméretű műveletnek, mert több mint két útvonaltervet képes módosítani. (A használt kis és nagy lépésméretű műveletek működésének a leírása a D. függelék D.3. szakaszában olvasható.) Ezeken kívül egy általam kifejlesztett újfajta közepes lépésméretű műveletet is alkal-

mazok, a részútvonalterv műveletet, amelyet ebben a munkában a 3.3. szakaszban mutatok be.

Az inverziós és a részútvonalterv műveletek csak a feladatok irányát és sorrendjét változtatják meg egy útvonalterven belül, így nem változtatják meg a megoldás megvalósíthatóságát. Ezzel szemben a betoldási, a csere és a kétopciós műveletek megváltoztathatják a kiszolgálandó igény mennyiségét egy vagy több útvonaltervben, így a megoldás megvalósíthatósága is megváltozhat. Emiatt – a beállításoktól függően – e műveletek kimeneti megoldása eltérő lehet. Ha a nem megvalósítható megoldások nem kerülnek elfogadásra, és a kiszámított kimeneti megoldás nem megvalósítható, akkor a művelet az eredeti, bemeneti megoldást adja vissza helyette (feltételezve, hogy a bemeneti megoldás megvalósítható).

3.2.5. Esemény és eseményfolyam

Az adat-vezérelt DCARP esetében a folyamatbányászatból már ismert eseményfolyamhoz (2.2.1. alszakasz) hasonló eseményfolyamot feltételezünk. Az eseményfolyam itt is megfigyelhető egységekből áll, ahol egy megfigyelhető egység egy eseménynek felel meg. Viszont itt egy eseményfolyam csak egy adott DCARP példára vonatkozó eseményeket tartalmaz, ezért az eseményfolyamot események véges sorozataként definiáljuk. Az eseményeket két nagy fő csoportba lehet osztani:

- **utazási vagy szolgáltatási események**: szolgáltatási folyamat végrehajtását leíró események,
- **úthálózathoz vagy járműhöz kapcsolódó események**: váratlanul bekövetkező, példát módosító események (lásd D. függelék D.1. szakasza).

Az egyszerűség kedvéért csak egy eseményfolyam kerül felhasználásra, ami az összes előforduló eseményt tartalmazza. Egy esemény minden eseményben előforduló attribútumot tartalmaz, de csak azoknál az attribútumoknál tartalmaz érvényes értéket, amik relevánsak az adott esemény szempontjából, a többinél a -1 érték szerepel. A megadott érték mindig új értéket jelent (pl. ha a dc_a értéke 10 az eseményben, akkor a $dc(a)$ új értéke 10 lesz az aktualizált DCARP példában). Egy eseményben használt attribútumok jelölése, jelentése és a lehetséges érvényes értékei a következők:

- ts: az esemény időbélyege, $ts \in \mathbb{Z}_{>0}$;
- a: az eseményhez kapcsolódó ív (útszakasz), $a \in A$;
- h: az eseményhez kapcsolódó jármű azonosítója, $h \in H$;
- α: az esemény típusa, $\alpha \in \{0, 1\}$ (a 0 érték utazási tevékenységet, az 1 pedig szolgáltatási tevékenységet jelöl);
- dc_a: az a ívhez tartozó új áthaladási költség, $dc_a \in \mathbb{R}_{\geq 0}$;
- sc_a: az a ívhez tartozó (új) szolgáltatási költség, $sc_a \in \mathbb{R}_{\geq 0}$;
- dem_a: az a ívhez tartozó (új) igény, $dem_a \in \mathbb{R}_{\geq 0}$;
- h_e: a h jármű állapotváltozását jelölő érték, $h_e \in \{0, 1\}$ (a 0 érték beüzemelést, az 1 pedig meghibásodást jelöl).

A lehetséges DCARP eseményeket és azt, hogy melyiknél mely attribútumok kerülnek megadásra, a 3.1. táblázat foglalja össze. A + jelölés azt jelzi, hogy az attribútum érvényes értéket kap, a − jelölés pedig azt jelzi, hogy az attribútum nem kap érvényes értéket (azaz -1 az értéke) az adott eseményben.

3.1. táblázat. A lehetséges DCARP események és azok attribútum értékei

Esemény	ts	a	h	α	dc_a	sc_a	dem_a	h_e
Utazás/szolgáltatás	+	+	+	+	-	-	-	-
Útlezárás	+	+	-	-	+	-	-	-
Útmegnyitás	+	+	-	-	+	-	-	-
Forgalom csökkenés	+	+	-	-	+	+	-	-
Forgalom növekedés	+	+	-	-	+	+	-	-
Feladat lemondás	+	+	-	-	-	-	-	-
Feladat megjelenés	+	+	-	-	-	+	+	-
Igény csökkenés	+	+	-	-	-	+	+	-
Igény növekedés	+	+	-	-	-	+	+	-
Jármű meghibásodás	+	-	+	-	-	-	-	+
Jármű beüzemelés	+	-	+	-	-	-	-	+

3.3. A javasolt részútvonalterv művelet

Ebben a fejezetben az általam a CARP-hoz kifejlesztett közepes lépésméretű mozgási művelet – a részútvonalterv művelet – kerül részletes bemutatásra. A használt jelölések a „Jelölések” között megtalálhatók.

3.3.1. Áttekintés

A részútvonalterv művelet egy közepes lépésméretű mozgási művelet, amelynek célja a CARP-hoz kifejlesztett evolúciós algoritmusok lokális keresési folyamatának hatékonyabbá tétele. A dolgozatomban a műveletet kizárólag a CARP-ABC algoritmusok (3.4. szakasz) használják, de bármely más CARP-hoz kifejlesztett evolúciós algoritmusban is alkalmazható.

A részútvonalterv művelet a TSP GSTM műveletén alapul [143]. A részútvonalterv és a GSTM műveletek közötti fő különbségek a TSP és a CARP közötti különbségekből adódnak. A részútvonalterv művelet csomópontok helyett ívekkel dolgozik. Továbbá, mivel egy TSP példa megoldása mindig egy útvonalterv, míg egy CARP példa megoldása (általában) egynél több útvonaltervből áll, ezért a részútvonalterv művelet a megoldásnak csak egy részét (egy útvonaltervet) vesz figyelembe a teljes megoldás helyett. Mivel csak egy útvonaltervet, és azon belül több feladat pozícióját módosíthatja, közepes lépésméretű műveletnek tekinthető.

A részútvonalterv művelet működését leíró fő algoritmus a 3.3.2. alszakaszban olvasható. Ez egy összetett mozgási művelet, amely két különböző mohó keresési módszert, a mohó újrakapcsolási módszert (3.3.3. alszakasz) és a részútvonalterv forgatási módszert (3.3.5. alszakasz), valamint egy torzítást biztosító módszert tartalmaz (3.3.4. alszakasz). A művelet mindhárom módszerben az érintett feladatok inverzióját is figyelembe veszi (amennyiben azok léteznek). A feladatok inverziójának akkor van igazán jelentősége, amikor a részútvonalterv forgatási módszerben egy feladat-sorozatot (azaz egy részútvonaltervet) invertál, mivel amikor a feladatok végrehajtási sorrendje megváltozik, akkor a feladatok végrehajtási irányát is érdemes megváltoztatni, hogy az utazási költség minimális maradjon.

3.3.2. A fő algoritmus

A részútvonalterv művelet fő algoritmusának algoritmikus leírása a 6. algoritmusban látható. Bemenetként az I CARP példát, annak egy S megoldását, valamint az algoritmus paramétereit várja. A paraméterek a következők:

- az újrakapcsolódás valószínűsége (p_{rc});
- a korrekció és a perturbáció valószínűsége (p_{cp});
- a linearitás valószínűsége (p_l);
- a részútvonalterv minimális hossza ($l_{\min}$);
- egy feladat figyelembe vett szomszédságának maximális mérete ($n_{\max}$).

Algoritmus 6: *subRoutePlan* (A részútvonalterv művelet algoritmusa)

```
   input  : I; S; p_rc, p_cp, p_l ∈ [0,1]; l_min, n_max ∈ ℕ⁺
 1 begin
 2     r_k ← selectRoutePlan(S);                          // útvonalterv kiválasztása a szolgáltatási tervből
 3     if l_k ≥ l_min then                                // ha elég hosszú az útvonalterv
 4         l_max ← max({l_min, toInt(√l_k)});             // a részútvonalterv maximális hosszának meghatározása
 5         r'_k ← r_k;                                    // az új útvonalterv inicializálása
 6         l ← randInt(l_min, l_max);                     // a részútvonalterv hosszának meghatározása
 7         s ← randInt(1, l_k − l + 1);                   // a kezdő feladatának pozícióindexének meghatározása
 8         e ← s + l − 1;                                 // a befejező feladatának pozícióindexének meghatározása
 9         r*_k ← ⟨t_{k,s}, ..., t_{k,e}⟩;                // a részútvonalterv előállítása
10         r#_k ← removeSubRoutePlan(r*_k, r_k);          // a részútvonalterv eltávolítása az útvonaltervből
11         p_rand ← randFloat(0, 1);                      // 0 és 1 közötti szám generálása
12         if p_rand ≤ p_rc then                          // ha a szám az újrakapcsolódás valószínűségénél nem nagyobb
13             r'_k ← greedyReconnection(I, r_k, r*_k, r#_k);   // a mohó újrakapcsolási módszer
                 alkalmazása
14         else
15             p_rand ← randFloat(0, 1);                  // 0 és 1 közötti szám generálása
16             if p_rand ≤ p_cp then                      // ha a szám a perturbáció valószínűségénél nem nagyobb
17                 r'_k ← distortion(I, r*_k, r#_k, s, p_l);   // a torzítási módszer alkalmazása
18             else
19                 r'_k ← subRoutePlanRotation(I, r_k, s, e, n_max);   // a részútvonalterv forgatási
                     módszer alkalmazása
20         S ← (S \ r_k) ∪ r'_k;                          // az útvonalterv frissítése a szolgáltatási tervben
21     return S;
```

Az algoritmus a részútvonalterv maximális hosszát ($l_{\max}$) az útvonalterv kiválasztása után határozza meg. A javasolt CARP-ABC algoritmusban az algoritmus paraméterei állandó értékként vannak megadva, így csak a CARP példát (I) és annak egy lehetséges megoldását (S) kell megadni.

Az algoritmus első lépésében kiválaszt egy r_k útvonaltervet az S megoldásból (2. sor), majd, ha az útvonalterven belüli (nem ál)feladatok száma elegendő (azaz l_k nagyobb vagy egyenlő $l_{\min}$, 3. sor), az algoritmus a következő lépésre megy tovább. Ellenkező esetben változatlanul visszaadja a bemeneti S szolgáltatási tervet (21. sor).

Az algoritmus következő lépésében a paraméterek inicializálása és a (rész)útvonaltervek generálása történik. A részútvonalterv maximális hosszát ($l_{\max}$) a kiválasztott r_k útvonalterven belüli feladatok száma (l_k) és a részútvonalterv előre meghatározott minimális hossza ($l_{\min}$) alapján meghatározza (4. sor), majd inicializálja az új útvonaltervet (r'_k) (5. sor). A részútvonalterv hosszát (l) véletlenszerűen határozza meg az $[l_{\min}, l_{\max}]$ intervallum alapján (6. sor). A részútvonalterv kezdő feladatának pozícióindexét (s) véletlenszerűen választja ki, figyelembe véve az l értékét (7. sor). A részútvonalterv befejező feladatának pozícióindexét (e) az s és az l értékek alapján határozza meg (8. sor). Az $r^\star_k$ részútvonaltervet úgy állítja elő, hogy az eredeti r_k útvonaltervből kiveszi azt a részútvonaltervet, amelyet a $t_{k,s}$ és $t_{k,e}$ feladatok határolnak (9. sor). Az $r^\star_k$ nélküli útvonaltervet $r^\#_k$ jelöli (10. sor).

Az algoritmus következő lépéseként egy „0” és „1” közötti véletlen számot (p_{rnd}) generál, amely meghatározza a művelet működését (11. sor). Ha p_{rnd} értéke kisebb vagy egyenlő az előre meghatározott p_{rc} újrakapcsolási valószínűséggel (azaz $p_{rnd} \leq p_{rc}$, 12. sor), akkor a mohó újrakapcsolási módszer kerül végrehajtásra (13. sor, 3.3.3. alszakasz), ellenkező esetben egy új véletlen szám kerül generálásra (15. sor). Ha a p_{rnd} új értéke kisebb vagy egyenlő az előre meghatározott p_{cp} korrekciós és perturbációs valószínűséggel (azaz, $p_{rnd} \leq p_{cp}$, 16. sor), akkor a torzítás adódik az r_k-hoz (17. sor, 3.3.4. alszakasz), ellenkező esetben a részútvonalterv forgatási módszerét hajtja végre (3.3.5. alszakasz). Utolsó lépésként az S szolgáltatási tervet frissíti a régi r_k útvonalterv eltávolításával és az új, r'_k útvonalterv hozzáadásával (20. sor), majd a frissített szolgáltatási tervet visszaadja (21. sor).

3.3.3. Mohó újrakapcsolási módszer

A mohó újrakapcsolási módszer a kiválasztott $r^\star_k$ részútvonaltervet az $r^\#_k$-n belül abba a pozícióba illeszti be, amely a legkevésbé növeli az útvonalterv teljes költségét.

Algoritmus

A mohó újrakapcsolási módszer algoritmikus leírása a részútvonalterv műveleten belül a 7. algoritmusban látható. Bemenetként az I CARP példát, az S megoldás eredeti r_k útvonaltervét, a kiválasztott $r^\star_k$ részútvonaltervet, és az $r^\#_k$ csonkított útvonaltervet (azaz r_k-t $r^\star_k$ nélkül) várja.

Algoritmus 7: *greedyReconnection* (A részútvonalterv művelet által használt mohó újrakapcsolási módszer algoritmusa)

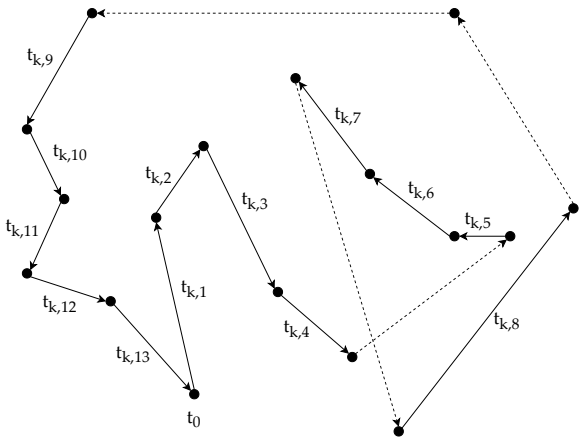
```
   input  : I; r_k; r*_k; r#_k
1  begin
2      r'_k ← r_k;                                  // az új útvonalterv inicializálása az aktuálissal
3      i ← 1;                                       // a pozícióindex inicializálása
4      while i ≤ l#_k + 1 do                        // a csonkított útvonalterven belüli pozíciók vizsgálata
5          r_{k,i} ← insertSubRoutePlan(r*_k, r#_k, i); // a részútvonalterv beszúrása a csonkított útvonalterv
             i pozíciójába
6          if TC({r_{k,i}}) < TC({r'_k}) then // ha a létrejött útvonalterv teljes költsége az eddigi legkisebb
7              r'_k ← r_{k,i};                      // az új útvonalterv lecserélése
8          i ← i + 1;
9      return r'_k;
```

Az algoritmus első lépésében az új r'_k útvonaltervet az aktuális r_k útvonaltervvel inicializálja (2. sor). Az i pozícióindexet is inicializálja, amely az $r^\star_k$ legjobb pozíciójának megtalálására szolgál az $r^\#_k$-ba való beilleszteshez (3. sor). Az „1” érték az $r^\#_k$ csonkított útvonalterven belüli első (nem ál)feladatra utal (azaz $t^\#_{k,1}$-ra). A „0” érték az első álfeladatra (azaz t_0-ra) és az $l^\#_k + 1$ pedig az utolsó álfeladatra utal az $r^\#_k$-on belül (feltételezve, hogy $l^\#_k$ a nem álfeladatok száma az $r^\#_k$-on belül). A következő lépésben az algoritmus az $r^\#_k$-on belüli minden egyes pozíciót megvizsgál, hogy megtalálja a legjobbat, ahová az $r^\star_k$-t be lehet illeszteni (4–8. sorok). Minden egyes iterációban az $r^\star_k$-t az *insertSubRoutePlan* függvénnyel illeszti be a $t^\#_{k,i}$ feladat elé (5. sor). Az így kapott $r_{k,i}$ útvonalterv összköltségét ezután összehasonlítja az r'_k útvonalterv összköltségével (6. sor). Ha $r_{k,i}$ jobb, mint r'_k (azaz kisebb a teljes költsége), akkor az lesz r'_k új értéke (7. sor). Az algoritmus utolsó lépésében az r'_k-t adja vissza a függvény (9. sor).

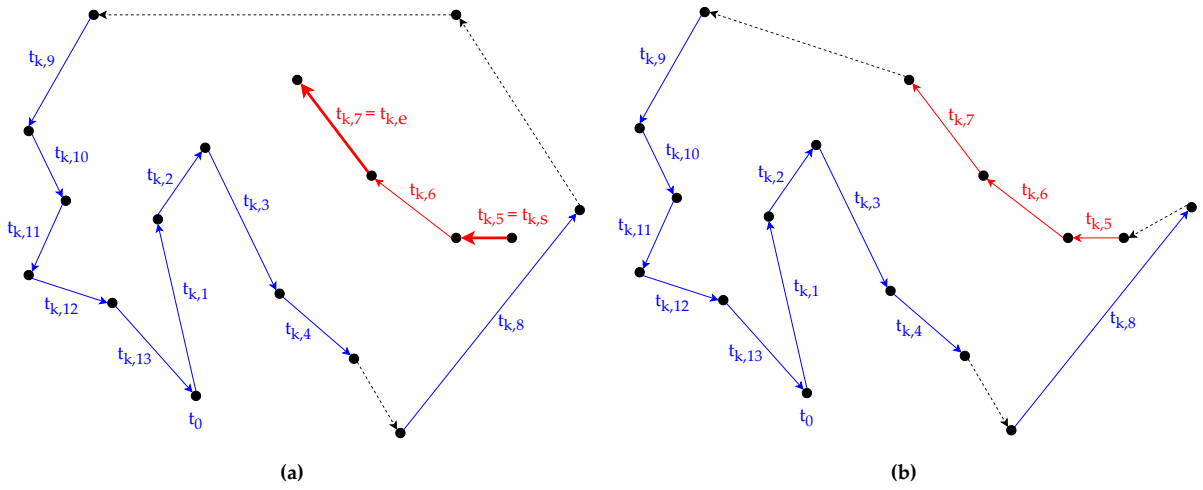
Példa

A módszer jobb megértéséhez tekintsük meg a következő példát. Legyen a kiválasztott útvonalterv $r_k = \langle t_0, t_{k,1}, t_{k,2}, \ldots, t_{k,13}, t_0 \rangle$ és a részútvonalterv hossza $l = 3$. Ezek alapján legyen a kiválasztott kezdő és befejező feladat $t_{k,s} = t_{k,5}$ és $t_{k,e} = t_{k,7}$, ekkor a kiválasztott részútvonalterv $r_k^\star = \langle t_{k,5}, t_{k,6}, t_{k,7} \rangle$ és az r_k útvonalterv $r_k^\star$ nélkül $r_k^\# = \langle t_0, t_{k,1}, t_{k,2}, t_{k,3}, t_{k,4}, t_{k,8}, t_{k,9}, t_{k,10}, t_{k,11}, t_{k,12}, t_{k,13}, t_0 \rangle$. Tegyük fel, hogy $r_k^\star$ beszúrása $t_{k,8}$ és $t_{k,9}$ közé $r_k^\#$-ban eredményezi a megoldás összköltségének legkisebb mértékű növekedését. Ekkor $r_k' = \langle t_0, t_{k,1}, t_{k,2}, t_{k,3}, t_{k,4}, t_{k,8}, t_{k,5}, t_{k,6}, t_{k,7}, t_{k,9}, t_{k,10}, t_{k,11}, t_{k,12}, t_{k,13}, t_0 \rangle$ lesz a megoldás új k-adik útvonalterve.

Az előző bekezdésben tárgyalt példát a 3.2. ábra és a 3.3. ábra mutatja be. Mindkét ábrán az útvonaltervben kiszolgált ívfeladatok folytonos vonallal, a többi, csak átjárt ívek pedig szaggatott vonallal vannak ábrázolva. A 3.2. ábrán az eredeti r_k útvonalterv, a 3.3. ábrán pedig az $r_k^\star$ részútvonalterv és a $r_k^\#$ csonka útvonalterv látható, piros, illetve kék színnel kiemelve. A kiválasztott kezdő és befejező feladatok ($t_{k,s}$ és $t_{k,e}$) vastagabb vonallal vannak jelölve.



3.2. ábra. Példa r_k útvonalterv.



3.3. ábra. Mohó újrakapcsolási módszer: (**a**) az $r_k^\star$ részútvonalterv lekapcsolva az r_k útvonaltervről; (**b**) az $r_k^\star$ részútvonalterv az $r_k^\#$ csonka útvonaltervhez kapcsolva.

3.3.4. Torzítási módszer

A torzítási módszer a kiválasztott $r_k^\star$ részútvonaltervben lévő feladatokat egyenként beilleszti az $r_k^\#$ csonkított útvonaltervbe, az s pozícióindexből kiindulva, és az előre meghatározott linearitási valószínűséggel (p_l) gördítve vagy keverve. A gördítés az aktuális utolsó feladat, a keverés pedig egy véletlenszerű feladat kiválasztását jelenti az $r_k^\star$-ban.

Algoritmus

A torzítási módszer algoritmikus leírása a részútvonalterv műveleten belül a 8. algoritmusban látható. Bemenetként az I CARP példát, a kiválasztott $r_k^\star$ részútvonaltervet, az $r_k^\#$ csonkított útvonaltervet (azaz r_k-t $r_k^\star$ nélkül), valamint a p_l linearitási valószínűséget várja.

Algoritmus 8: *distortion* (A részútvonalterv művelet által használt torzítási módszer algoritmusa)

```
   input  : I; r_k^⋆; r_k^#; s; p_l
1  begin
2      r'_k ← r_k^#;                          // az új útvonalterv inicializálása a csonkított útvonaltervvel
3      i ← s;                                 // a pozícióindex inicializálása a kezdő feladat pozícióindexével
4      while |r_k^⋆| ≠ 0 do                   // amíg van még beszúrandó feladat
5          t ← t^⋆_{k,|r_k^⋆|};               // a részútvonalterv utolsó feladatának kiválasztása
6          p_rnd ← randFloat(0, 1);           // 0 és 1 közötti szám generálása
7          if p_rnd ≤ p_l then                // ha a szám nem nagyobb a linearitási valószínűségnél
8              j ← randInt(1, |r_k^⋆|);       // véletlenszerű pozícióindex kiválasztása a részútvonalterven
9              t ← t^⋆_{k,j};                 // az adott pozícióban lévő feladat kiválasztása
10         r'_k ← insertTask(t, r'_k, i);     // a feladat beszúrása az új útvonaltervbe
11         r_k^⋆ ← removeTask(t, r_k^⋆);      // a feladat eltávolítása a részútvonaltervből
12         i ← i + 1;
13     return r'_k;
```

Az algoritmus első lépésében az r'_k új útvonaltervet az $r_k^\#$ csonkított útvonaltervvel inicializálja (2. sor), és az i pozícióindexet s értékkel inicializálja (3. sor). Amíg vannak feladatok a kiválasztott $r_k^\star$ részútvonaltervben, az algoritmus a következő lépéseket hajtja végre (4–12. sorok). Először is kiválasztja az utolsó feladatot az $r_k^\star$-ból és a t változóba menti (5. sor), majd generál egy véletlen számot (p_{rnd}) „0” és „1” érték között (6. sor). Ha p_{rnd} értéke kisebb vagy egyenlő, mint p_l értéke, akkor t értékét egy véletlenszerű feladatra cseréli $r_k^\star$-ból (7–9. sorok). A kiválasztott t feladatot ezután az r'_k-ban az $insertTask$ függvénnyel (10. sor) beilleszti az i pozícióba, és a $removeTask$ függvénnyel (11. sor) eltávolítja az $r_k^\star$-ból. A t feladat mindig közvetlenül a korábban beillesztett feladat után kerül beillesztésre. Amikor az $r_k^\star$ kifogyott a feladatokból (azaz az összes benne lévő feladat beillesztésre került az r'_k-ba), az algoritmus visszaadja r'_k-t (13. sor).

Példa

A módszer jobb megértéséhez tekintsük meg a következő példát. Legyen a kiválasztott útvonalterv $r_k = \langle t_0, t_{k,1}, t_{k,2}, \ldots, t_{k,13}, t_0 \rangle$ és a részútvonalterv hossza $l = 3$. Ezek alapján legyen a kiválasztott kezdő és befejező feladat $t_{k,s} = t_{k,5}$ és $t_{k,e} = t_{k,7}$, ekkor a kiválasztott részútvonalterv $r_k^\star = \langle t_{k,5}, t_{k,6}, t_{k,7} \rangle$ és az r_k útvonalterv $r_k^\star$ nélkül $r_k^\# = \langle t_0, t_{k,1}, t_{k,2}, t_{k,3}, t_{k,4}, t_{k,8}, t_{k,9}, t_{k,10}, t_{k,11}, t_{k,12}, t_{k,13}, t_0 \rangle$. Az $r_k^\star$ hossza három,

tehát ebben az esetben az algoritmusnak három iterációja van. Legyen a linearitás valószínűsége $p_l = 0,2$.

Az első iterációban a véletlen szám legyen $p_{rnd} = 0,1$. Mivel ez kisebb, mint p_l, egy véletlenszerű feladat kerül kiválasztásra $r_k^\star$-ból. Legyen ez a kiválasztott feladat $t_{k,6}$. Ebben az esetben az új útvonalterv $r'_k = \langle t_0, t_{k,1}, t_{k,2}, t_{k,3}, t_{k,4}, t_{k,6}, t_{k,8}, \ldots \rangle$ (azaz a $t_{k,6}$ feladat a $t_{k,4}$ és $t_{k,8}$ feladatok közé kerül) és $r_k^\star = \langle t_{k,5}, t_{k,7} \rangle$ (azaz a $t_{k,6}$ feladat eltávolításra kerül).

A második iterációban a véletlen szám legyen $p_{rnd} = 0,8$. Mivel ez nagyobb, mint p_l, az utolsó feladat kerül kiválasztásra $r_k^\star$-ból, ami $t_{k,7}$. Ebben az esetben az új útvonalterv $r'_k = \langle t_0, t_{k,1}, t_{k,2}, t_{k,3}, t_{k,4}, t_{k,6}, t_{k,7}, t_{k,8}, \ldots \rangle$ (azaz a $t_{k,7}$ feladat $t_{k,6}$ és $t_{k,8}$ feladatok közé kerül) és $r_k^\star = \langle t_{k,5} \rangle$.

A harmadik iterációban, mivel csak egy feladat maradt $r_k^\star$-ban, a p_{rnd} értékétől függetlenül a $t_{k,5}$ feladat kerül kiválasztásra. Így az új útvonalterv $r'_k = \langle t_0, t_{k,1}, t_{k,2}, t_{k,3}, t_{k,4}, t_{k,6}, t_{k,7}, t_{k,5}, t_{k,8}, \ldots \rangle$ és $r_k^\star = \langle \rangle$. Mivel $r_k^\star$ már üres, az algoritmus r'_k-t visszaadja.

3.3.5. Részútvonalterv forgatási módszer

A részútvonalterv forgatási módszer véletlenszerűen kiválaszt egy szomszédos feladatot a $t_{k,s}$ és $t_{k,e}$ feladatok szomszédai közül (azaz $t_{k,s^\star}$ és $t_{k,e^\star}$ feladatokat), majd megfordítja a $t_{k,i}$ és a $t_{k,i^\star}$ feladatok közötti részútvonaltervet (beleértve a $t_{k,i^\star}$ feladatot), ahol $(i, i^\star) \in \{(s, s^\star), (e, e^\star)\}$. A sorozat megfordítása úgy történik, hogy $t_{k,i}$ és $t_{k,i^\star}$ (vagy $inv(t_{k,i^\star})$) közvetlen szomszédok lesznek az r'_k új útvonaltervben.

Algoritmus

A részútvonalterv forgatási módszer algoritmikus leírása a részútvonalterv műveleten belül a 9. algoritmusban látható. Bemenetként az I CARP példát, az S megoldás eredeti r_k útvonaltervét, a kezdő feladat s és a végfeladat e pozícióindexét (az eredeti r_k útvonalterven belül, ami meghatározza az $r_k^\star$ részútvonaltervet), valamint az $n_{\max}$ figyelembe vett (maximális) szomszédság méretet várja. Az $n_{\max}$ értéke határozza meg, hogy egy feladat esetén maximum hány szomszédos (legközelebbi) feladatot veszünk figyelembe a szomszédos feladat kiválasztásakor.

Az algoritmus első lépésében az $n_{\max}$ legközelebbi szomszédos feladatok közül véletlenszerűen kiválaszt egy-egy pozícióindexet a $t_{k,s}$ és a $t_{k,e}$ feladatok számára ($s^\star$ és $e^\star$, rendre) a $selectNeighborTaskIndex$ függvénnyel (2–3. sorok). Továbbá, az r'_k új útvonalterv inicializálásra kerül az r_k eredeti útvonaltervvel (4. sor). A következő lépésben minden $(i, i^\star) \in \{(s, s^\star), (e, e^\star)\}$ indexre a következő lépéseket hajtja végre (5–22. sorok). Először az $r'_{k,i}$ potenciális új útvonaltervet egy üres útvonaltervvel inicializálja (6. sor), majd az i és $i^\star$ index értékek közötti kapcsolat alapján kiválaszt egy alútvonaltervet és megfordítja azt. Ha az eredeti r_k útvonaltervben a $t_{k,i}$ feladat a $t_{k,i^\star}$ feladat előtt van (azaz $i < i^\star$), akkor az r'_k új útvonaltervben a $t_{k,i}$ feladat után közvetlenül a $t_{k,i^\star}$ (vagy $inv(t_{k,i^\star})$) feladat lesz (7–8. sorok). Ellenkező esetben, az eredeti r_k útvonaltervben a $t_{k,i^\star}$ feladat a $t_{k,i}$ feladat előtt van, akkor az r'_k új útvonaltervben a $t_{k,i^\star}$ (vagy az $inv(t_{k,i^\star})$) feladat után közvetlenül a $t_{k,i}$ feladat lesz (14–15. sorok). Mindkét esetben minden olyan feladatot, amelynek van inverze, és amely a megfordított részútvonalterven belül van, az r'_k új útvonaltervben a $replaceTaskInv$ függvénnyel az inverz feladatra cseréli (9–13. sorok és 16–20. sorok).

A részútvonaltervek közül a legalacsonyabb összköltséggel rendelkezőt válassza ki (21–22. sorok) és adja vissza (23. sor).

Algoritmus 9: *subRoutePlanRotation* (A részútvonalterv művelet által használt részútvonalterv forgatási módszer algoritmusa)

```
   input  : I; r_k; s; e; n_max
1  begin
2      s* ← selectNeighborTaskIndex(r_k, s, n_max);        // feladat pozícióindex kiválasztása t_{k,s} környezetében
3      e* ← selectNeighborTaskIndex(r_k, e, n_max);        // feladat pozícióindex kiválasztása t_{k,e} környezetében
4      r'_k ← r_k;                                         // az új útvonalterv inicializálása az aktuálissal
5      forall (i, i*) ∈ {(s, s*), (e, e*)} do
6          r'_{k,i} ← ⟨⟩;                                  // a potenciális új útvonalterv inicializálása
7          if i < i* then                                  // ha a t_{k,i} feladat t_{k,i*} feladat előtt van
8              r'_{k,i} ← ⟨t_0, t_{k,1}, ..., t_{k,i}⟩ · ⟨t_{k,i*}, t_{k,i*-1}, ..., t_{k,i+1}⟩ · ⟨t_{k,i*+1}, ..., t_{k,l_k}, t_0⟩;   // a t_{k,i+1} és t_{k,i*} feladatok által határolt részútvonalterv megfordítása az útvonaltervben
9              j ← i*;
10             while i < j do                              // minden érintett feladat esetében
11                 if ∃inv(t_{k,j}) then                   // ha a feladatnak létezik inverze
12                     r'_{k,i} ← replaceTaskInv(r'_{k,i}, t_{k,j}, inv(t_{k,j}));   // a feladat lecserélése annak inverzére
13                 j ← j − 1;
14         else
15             r'_{k,i} ← ⟨t_0, t_{k,1}, ..., t_{k,i*-1}⟩ · ⟨t_{k,i-1}, t_{k,i-2}, ..., t_{k,i*}⟩ · ⟨t_{k,i}, ..., t_{k,l_k}, t_0⟩;   // a t_{k,i*} és t_{k,i-1} feladatok által határolt részútvonalterv megfordítása az útvonaltervben
16             j ← i − 1;
17             while i* ≤ j do                             // minden érintett feladat esetében
18                 if ∃inv(t_{k,j}) then                   // ha a feladatnak létezik inverze
19                     r'_{k,i} ← replaceTaskInv(r'_{k,i}, t_{k,j}, inv(t_{k,j}));   // a feladat lecserélése annak inverzére
20                 j ← j − 1;
21         if TC({r'_{k,i}}) < TC({r'_k}) then // ha a potenciális új útvonalterv jobb, mint az új útvonalterv
22             r'_k ← r'_{k,i};                            // az új útvonalterv lecserélése
23     return r'_k;
```

A szomszédság meghatározása

A szomszédok meghatározása a szomszédság előre meghatározott maximális mérete ($n_{\max}$) alapján történik. A szomszédságba itt a nem közvetlen szomszédok is beletartoznak. Ha a szomszédság maximális mérete $n_{\max}$, akkor az algoritmus a $t_{k,i}$ ($i \in \{s, e\}$) feladathoz legközelebb lévő, maximum $n_{\max}$ darab feladatot fogja figyelembe venni az r_k útvonaltervben.

A $t_{k,i}$ feladat és egy másik feladat közötti távolságot két dolog határoz meg: a köztük lévő legrövidebb útvonal hossza, valamint az, hogy a másik feladatnak van-e inverz feladata (azaz egy élfeladattól származik-e) vagy sem. Legyen $t_{k,j}$ egy olyan feladat az r_k útvonalterven, amely nem azonos $t_{k,i}$ feladattal (azaz $t_{k,i} \neq t_{k,j}$). Ha a $t_{k,j}$ feladat az r_k útvonalterven $t_{k,i}$ előtt van (azaz $j < i$) és van inverz feladata (azaz $inv(t_{k,j}) \in T$), akkor a két feladat közötti távolság a fejcsúcsaik közötti legrövidebb út, ami az $mdc\Big(head(t_{k,j}), head(t_{k,i})\Big)$ kifejezéssel kiszámítható. A legrövidebb utat a $head(t_{k,j})$ fejcsúcsból kiindulva számítjuk ki, mivel a részútvonalterv forgatása során a $t_{k,j}$ feladat megfordul (azaz az útvonaltervben az inverzére cserélődik), és tudjuk, hogy egy feladat fejcsúcsa megegyezik az inverz feladatának a végcsúcsával (azaz $head(t_{k,j}) = tail\Big(inv(t_{k,j})\Big)$). Ha a feladatnak nincs inverze, akkor a $tail(t_{k,j})$ végcsúcsból kiindulva számoljuk ki a legrövidebb utat. Ha a $t_{k,j}$ feladat a $t_{k,i}$ feladat után van (azaz $j > i$) és van inverz feladata, akkor az $mdc\Big(tail(t_{k,i}), tail(t_{k,j})\Big)$

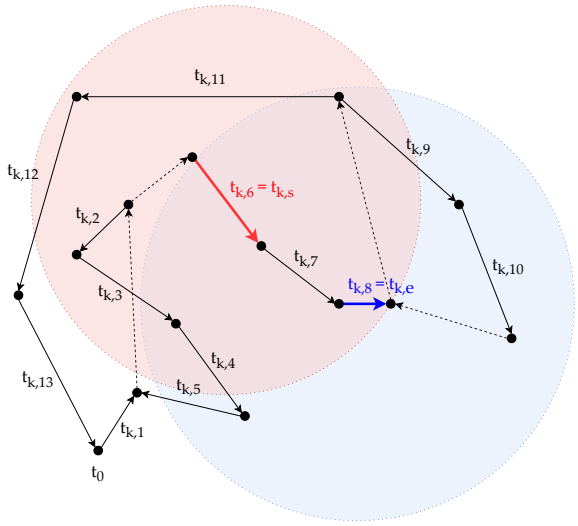
kifejezés kerül kiszámításra. Ha a feladatnak nincs inverze, akkor a $head(t_{k,j})$ fejcsúcsnál végződő legrövidebb út kerül kiszámításra. A 3.2. táblázat összefoglalja, hogy milyen kifejezést használjunk a különböző esetekben.

3.2. táblázat. Összefoglaló táblázat arról, hogy milyen kifejezést kell használni a $t_{k,j}$ és a $t_{k,i}$ $(i \in \{s, e\})$ feladatok közötti távolság kiszámításához az r_k útvonaltervben.

A $t_{k,j}$ a $t_{k,i}$ előtt vagy után van az r_k-ban?	**Van-e $t_{k,j}$-nek inverz feladata?**	**Kifejezés**
előtt	van	$mdc\Big(head(t_{k,j}), head(t_{k,i})\Big)$
	nincs	$mdc\Big(tail(t_{k,j}), head(t_{k,i})\Big)$
után	van	$mdc\Big(tail(t_{k,i}), tail(t_{k,j})\Big)$
	nincs	$mdc\Big(tail(t_{k,i}), head(t_{k,j})\Big)$

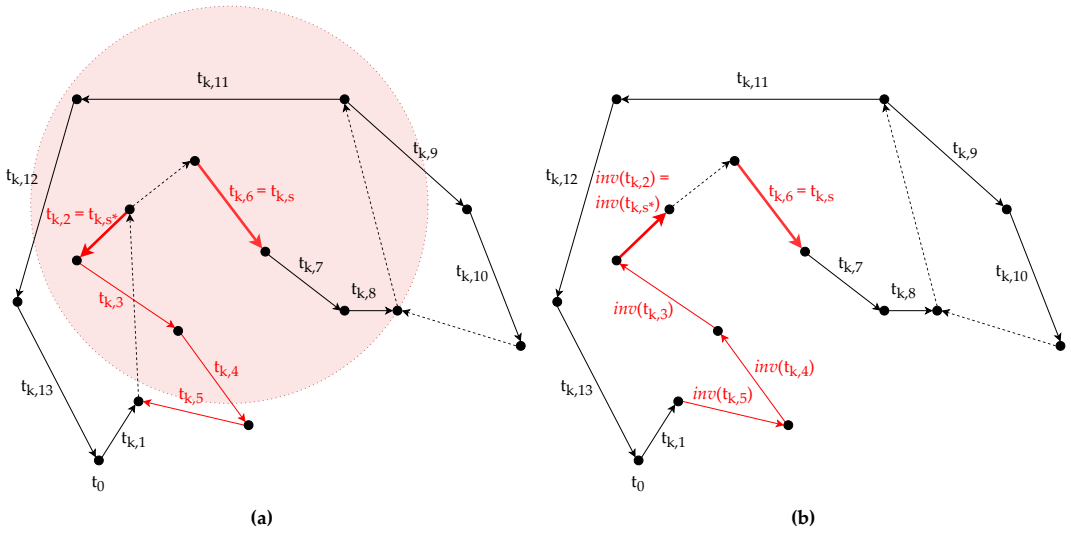
Példa

A módszer jobb megértéséhez tekintsük meg a következő példát. Legyen a kiválasztott útvonalterv $r_k = \langle t_0, t_{k,1}, t_{k,2}, \ldots, t_{k,13}, t_0 \rangle$, a részútvonalterv hossza $l = 3$ és a vizsgált szomszédság maximális mérete $n_{\max} = 5$. (Ennél a módszernél az l értéke csak a két kiválasztott feladat közötti távolságot határozza meg, nincs hatással az elforgatott részútvonaltervek hosszára.) Ezek alapján legyen a kiválasztott két feladat $t_{k,s} = t_{k,6}$ és $t_{k,e} = t_{k,8}$. Tegyük fel, hogy az r_k útvonalterv összes feladata élfeladatból származik, tehát mindegyiknek van inverz feladata.

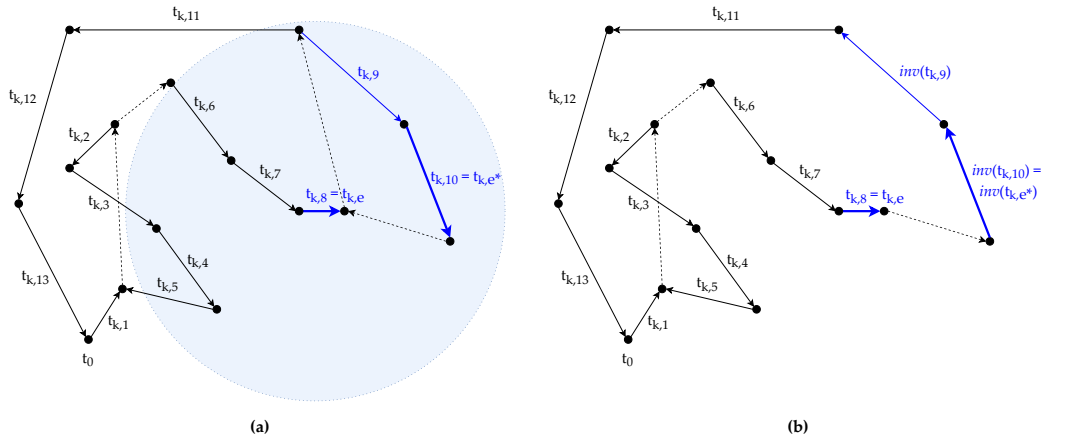


3.4. ábra. A $t_{k,s} = t_{k,6}$ és $t_{k,e} = t_{k,8}$ feladatok legközelebbi szomszédai az r_k útvonalterven belül, ha $n_{\max} = 5$.

Az útvonaltervet, a kiválasztott feladatokat és azok szomszédait a 3.4. ábra szemlélteti. Az útvonaltervben kiszolgált feladatok folytonos vonalakkal, míg a csak áthaladásra használt ívek szaggatott vonalakkal vannak jelölve. A $t_{k,s}$ és a $t_{k,e}$ feladatok



3.5. ábra. A $t_{k,s^\star}$ és a $t_{k,s}$ feladatok által bezárt részútvonalterv elforgatása: (**a**) A $t_{k,s}$ (itt $t_{k,6}$) feladat szomszédjai közül véletlenszerűen kiválasztunk egy $t_{k,s^\star}$ (itt $t_{k,2}$) feladatot, így kapunk egy részútvonaltervet; (**b**) A részútvonaltervet megfordítjuk, így az $inv(t_{k,s^\star})$ inverz feladatot közvetlenül a $t_{k,s}$ feladat fogja követni az r'_k új útvonaltervben.



3.6. ábra. A $t_{k,e}$ és a $t_{k,e^\star}$ feladatok által bezárt részútvonalterv elforgatása: (**a**) A $t_{k,e}$ (itt $t_{k,8}$) feladat szomszédjai közül véletlenszerűen kiválasztunk egy $t_{k,e^\star}$ (itt $t_{k,10}$) feladatot, így kapunk egy részútvonaltervet; (**b**) A részútvonaltervet megfordítjuk, így a $t_{k,e}$ feladatot közvetlenül az $inv(t_{k,e^\star})$ inverz feladat fogja követni az r'_k új útvonaltervben.

és azok szomszédsága piros, illetve kék színnel vannak kiemelve. Csak azok a feladatok tartoznak a szomszédságba, amelyeket az ellipszisek teljesen lefednek. Fontos megjegyezni, hogy az ellipszisek csak ábrázolási célt szolgálnak. Mivel a szomszédság azonosítása a távolságszámítás miatt meglehetősen bonyolult, ez azt eredményezi, hogy a csak szomszédos feladatokat lefedő alakzat formája változó. Ezek alapján a $t_{k,s}$ feladat esetén a szomszédos feladatok halmaza $\{t_{k,2}, t_{k,3}, t_{k,7}, t_{k,8}, t_{k,11}\}$, míg a $t_{k,e}$ feladat esetén $\{t_{k,4}, t_{k,6}, t_{k,7}, t_{k,9}, t_{k,10}\}$.

A $t_{k,s}$ feladat esetében tegyük fel, hogy a kiválasztott szomszédos feladat $t_{k,2}$ (azaz $t_{k,s^\star} = t_{k,2}$), ekkor a részútvonalterv $r_k^\star = \langle t_{k,2}, t_{k,3}, t_{k,4}, t_{k,5}\rangle$ (3.5a. ábra). Mivel a $t_{k,s^\star}$ feladat megelőzi a $t_{k,s}$ feladatot (azaz $s^\star < s$), a részútvonaltervet úgy fordítjuk meg, hogy az új útvonaltervben $inv(t_{k,s^\star})$ inverz feladatot közvetlenül a $t_{k,s}$ feladat követi. A megfordított részútvonalterv $\langle inv(t_{k,5}), inv(t_{k,4}), inv(t_{k,3}), inv(t_{k,2})\rangle$, így az új útvonalterv $r'_k = \langle t_0, t_{k,1}, inv(t_{k,5}), inv(t_{k,4}), inv(t_{k,3}), inv(t_{k,2}), t_{k,6}, \ldots, t_{k,13}, t_0\rangle$ (3.5b. ábra).

A $t_{k,e}$ feladat esetében tegyük fel, hogy a kiválasztott szomszédos feladat $t_{k,10}$ (azaz $t_{k,e^\star} = t_{k,10}$), ekkor a részútvonalterv $r_k^\star = \langle t_{k,9}, t_{k,10}\rangle$ (3.6a. ábra). Mivel a $t_{k,e^\star}$ feladat a $t_{k,s}$ feladat után van (azaz $e < e^\star$), a részútvonaltervet úgy fordítjuk meg, hogy az új útvonaltervben $inv(t_{k,e^\star})$ inverz feladat közvetlenül követi a $t_{k,e}$ feladatot. A megfordított részútvonalterv $\langle inv(t_{k,10}), inv(t_{k,9})\rangle$, így az új útvonalterv $r'_k = \langle t_0, t_{k,1}, \ldots, t_{k,8}, inv(t_{k,10}), inv(t_{k,9}), t_{k,11}, \ldots, t_{k,13}, t_0\rangle$ (3.6b. ábra).

3.4. A javasolt CARP-ABC algoritmusok

Ebben a szakaszban az általam a CARP-hoz kifejlesztett ABC algoritmusok – a CARP-ABC algoritmusok – kerülnek bemutatásra. A CARP-ABC algoritmusokhoz használt jelölések a „Jelölések” között megtalálhatók.

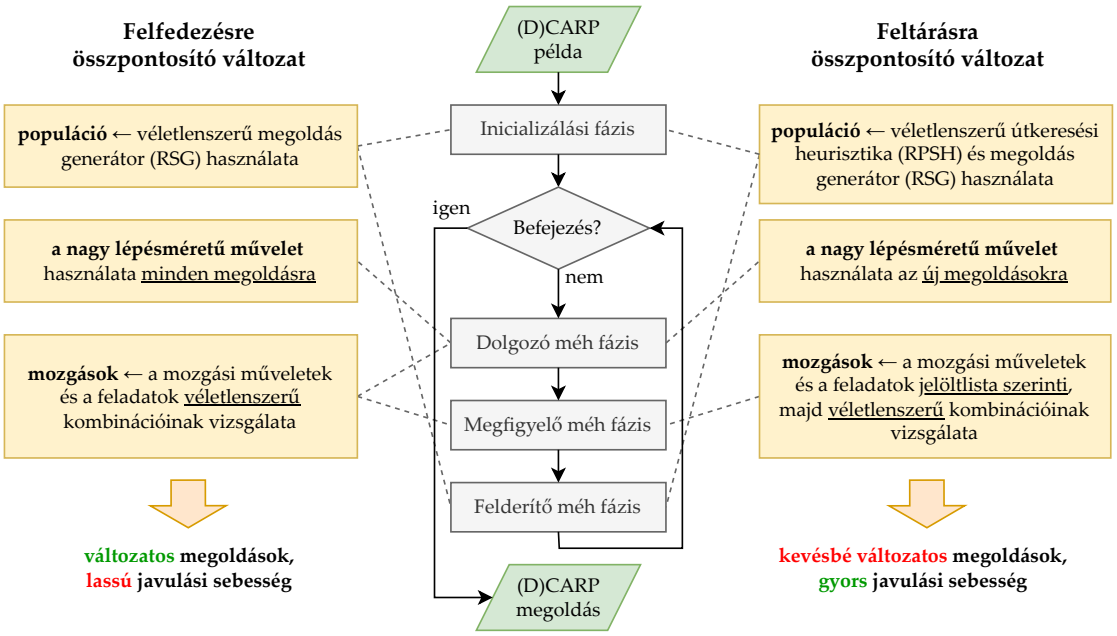
3.4.1. Áttekintés

A javasolt CARP-ABC algoritmusból két változat készült:

- **Felfedezésre összpontosító változat**: Ennek a változatnak a célja, hogy minél változatosabb megoldásokat biztosítson annak érdekében, hogy minél alaposabb keresést végezzen a megoldástérben. A keresési folyamatot többnyire véletlenszerű döntések irányítják; a populációt véletlenszerű megoldásokkal inicializálja és frissíti, valamint a keresési folyamat során a vizsgált lépéseket is véletlenszerűen választja ki. Mivel inkább a felfedezésre összpontosít, ezért a legfőbb hátránya, hogy lassan javul az általa talált globálisan legjobb megoldás minősége. Ebből adódóan CARP megoldóként javasolt használni, mivel ott nem olyan fontos a gyorsaság.

- **Feltárásra összpontosító változat**: Ennek a változatnak a célja, hogy minél hamarabb találjon minél jobb megoldást. A keresési folyamatot véletlenszerű döntések helyett inkább tudatos döntések vezérlik. A populációt finomított megoldásokkal inicializálja és frissíti, valamint a keresési folyamat során a vizsgált lépéseket a potenciális minőségjavító hatásuk alapján választja ki. Mivel inkább a gyors javításra összpontosít, ezért a legfőbb hátránya, hogy nagyobb valószínűséggel ad egy lokálisan optimális megoldást, mint a globálisan legjobb

megoldást. Ebből adódóan DCARP megoldóként javasolt használni, mivel ott fontosabb a gyorsaság.



3.7. ábra. A javasolt CARP-ABC algoritmus két változata

A CARP-ABC algoritmus fő struktúrája és a két változat közötti különbségek a 3.7. ábrán láthatók. A felfedezésre összpontosító CARP-ABC algoritmus működését leíró fő algoritmus a 3.4.2. alszakaszban található és a számítási komplexitása a D. függelék D.4. szakaszában kerül elemzésre. Az alap ABC algoritmushoz hasonlóan négy fázisból áll: az inicializálási fázissal (3.4.3. alszakasz) indul, majd három méhfázis ismétlődik mindaddig, amíg az előre meghatározott befejezési kritérium nem teljesül. Ez a három fázis a dolgozó méh fázis (3.4.4. alszakasz), a megfigyelő méh fázis (3.4.5. alszakasz) és a felderítő méh fázis (3.4.6. alszakasz). A feltárásra összpontosító CARP-ABC algoritmus működése a felfedezésre összpontosító változathoz viszonyítva kerül bemutatásra a 3.4.7. alszakaszban.

3.4.2. A fő algoritmus

A CARP-ABC algoritmus algoritmikus leírása a 10. algoritmusban látható. Az algoritmus négy fő fázisra osztható: inicializálási, dolgozó méh, megfigyelő méh és felderítő méh fázisokra. Az algoritmus az inicializálási fázissal kezdődik, majd egy ciklusba lép, ahol az említett fázisokat a megfelelő sorrendben addig ismétli, amíg a befejezési kritérium nem teljesül (4. sor). Az inicializálási fázisban (2. sor) kerül inicializálásra a kolónia (C), a kolónián belüli megoldások kora ($\mathcal{A}$), a globálisan legjobb megoldás ($S^\star$) és annak kora ($\alpha^\star$). Az algoritmus a dolgozó méh fázisban (6. sor) a kolónia tagjai körül lokális keresést végez. A megfigyelő méh fázisban (7. sor) pedig a kolónia egy megoldása körül hajt végre egy alaposabb lokális keresést. A felderítő méh fázisban globális keresés valósul meg (8. sor).

Az algoritmus paraméterei a következők:

- I: egy CARP példa;
- n_{cs}: a kolónia mérete, a populációban lévő megoldások száma;

- n_{mi}: az algoritmus maximális iterációinak száma;
- n_{gsl}: a globális keresési limit, az egymást követő iterációk maximálisan megengedett száma, amelyben az aktuálisan ismert globálisan legjobb megoldás nem javul;
- n_{lsl}: a lokális keresési limit, az algoritmus dolgozó méh fázisában és a megfigyelő méh fázisában az egymást követő iterációk maximálisan megengedett száma, amelyben az aktuálisan ismert lokálisan legjobb megoldás nem javul;
- n_{sal}: a megoldás korhatára, az algoritmus azon egymást követő iterációinak maximálisan megengedett száma, amelyben egy megoldás a populációban marad;
- egy befejezési kritérium, amely alapesetben az, hogy vagy az n_{mi} vagy az n_{gsl} érték elérésre kerül.

Algoritmus 10: *CARPABC* (A felfedezésre összpontosító CARP-ABC algoritmus)

```
   input  : I; n_cs, n_mi, n_gsl, n_lsl, n_sal ∈ ℕ
1  begin
2      C, A, S*, α* ← initializationP(I, n_cs);          // inicializálási fázis
3      i ← 0;                                  // az iteráció számláló inicializálása
4      while i < n_mi ∧ α* < n_gsl do  // iteráció, amíg az iterációs és a globális keresési limitet el nem éri
5          C̄ ← C;                                  // az előző kolónia elmentése
6          C, P ← employedBP(I, C, n_lsl);          // dolgozó méh fázis
7          C ← onlookerBP(I, C, P, n_cs, n_lsl);    // megfigyelő méh fázis
8          C, A, S*, α* ← scoutBP(I, C̄, C, A, S*, α*, n_sal);   // felderítő méh fázis
9          i ← i + 1;                              // az iteráció számláló növelése
10     return S*;
```

3.4.3. Inicializálási fázis

A CARP-ABC algoritmus inicializálási fázisának algoritmikus leírása a 11. algoritmusban látható. Az algoritmus ebben a fázisban először minden S_i ($i = 1, 2, \ldots, n_{cs}$) megoldásokhoz inicializálja a halmazokat, illetve azok korát (2–3. sorok). A jó minőségű és változatos kezdeti populáció garantálása érdekében a megoldásokat véletlenszerűen generálja a CARP-hoz kifejlesztett véletlenszerű megoldás generátor (*Random Solution Generation*, röviden *RSG*) algoritmus segítségével (6. sor), amely a következő alszakaszban kerül bemutatásra. Más populáció alapú evolúciós algoritmusok általában a véletlenszerű útkeresési heurisztikát (*Randomized Path-Scanning Heuristic*, röviden *RPSH*) [118] módszert használják a kezdeti megoldások generálására. A vizsgálatok során azonban azt tapasztaltam, hogy a CARP-ABC algoritmus esetében ez lényegesen nem javítja az algoritmus konvergenciájának a sebességét, ezért csak az RSG-t használtam.

A kolónia inicializálása után az algoritmus a *selectBestSolution* függvény segítségével kiválasztja a legjobb (legmagasabb) megfelelőségi értékkel rendelkező $S_i \in C$ megoldást (11. sor). Egy S_i megoldás megfelelőségét a teljes költsége $TC(S_i)$ határozza meg (ennek a lehető legkisebbnek kell lennie). Ennélfogva egy S_i megoldás megfelelőségi értékét a következő függvénnyel számoljuk ki:

$$fit(S_i) = \frac{LB(I)}{TC(S_i)} \tag{3.13}$$

ahol $LB(I)$ az I CARP példa megoldásának teljes költségének alsó korlátja, ami az összes feladat szolgáltatási költségének az összege (amennyiben egy feladat élfeladat,

annak csak az egyik irányát számítja bele). A megfelelőségi érték „0” és „1” között mozog. A nagyobb megfelelőségi értékkel rendelkező megoldások előnyben részesülnek, mivel a nagyobb érték azt jelenti, hogy a megoldás összköltsége közelebb van az alsó korláthoz.

Algoritmus 11: *initializationP* (A felfedezésre összpontosító CARP-ABC algoritmus inicializálási fázisa)

```
   input  : I; n_cs ∈ ℕ
1  begin
2      C ← ∅;                                          // kolónia halmaz inicializálása
3      A ← ∅;                                          // (kolóniához tartozó) korok halmazának inicializálása
4      i ← 1;                                          // iteráció számláló inicializálása
5      while i ≤ n_cs do                               // amíg a megoldások száma nem egyenlő a kolónia méretével
6          S_i ← RSG(I);                               // egy lehetséges megoldás létrehozása az RSG algoritmussal
7          α_i ← 0;                                    // a megoldás korának inicializálása
8          C ← C ∪ S_i;                                // a megoldás hozzáadása a kolóniához
9          A ← A ∪ α_i;        // a megoldás korának hozzáadása a (kolóniához tartozó) korok halmazához
10         i ← i + 1;                                  // iteráció számláló növelése
11     S* ← selectBestSolution(C);                     // a legjobb megoldás kiválasztása a kolóniából
12     α* ← 0;                                         // a legjobb megoldás korának inicializálása
13     return C, A, S*, α*;
```

Az RSG algoritmus

Első lépésként az algoritmus n_{cs} számú véletlenszerű permutációt generál, amelyek „1” és n közötti pozitív egész számokat tartalmaznak (azaz minden ívfeladat azonosítóját és minden élfeladat két *id*-jének egyikét) véletlenszerű sorrendben. Következő lépésként az algoritmus a permutációban szereplő *id*-ket balról jobbra haladva egyenként kiolvassa, miközben a hozzájuk tartozó feladatok igényét összeadja. Ha az aktuálisan beolvasott *id*-hez tartozó feladat megszegné az aktuális útvonalterv kapacitáskorlátozását, az algoritmus a feladat *id*-je elé egy „0”-t (a t_0 álfeladat *id*-jét) illeszt (azaz a feladatot egy új útvonaltervhez adja hozzá). Miután befejezte a feladatok *id*-inek útvonaltervekbe való felosztását, az algoritmus minden egyes feladatot végignéz a megoldásban, és ha van inverz feladata, akkor véletlenszerűen (pl. 0,5 valószínűséggel) kicseréli a feladat *id*-jét az inverzének *id*-jével. Utolsó lépésként az algoritmus a megoldás első és utolsó feladataként az álfeladatot (azaz „0”-t) illeszti be, hogy a megoldás érvényes megoldás legyen.

3.4.4. Dolgozó méh fázis

A CARP-ABC algoritmus dolgozó méh fázisának algoritmikus leírása a 12. algoritmusban látható. Az algoritmus ebben a fázisban minden egyes dolgozó méh esetében az ebben a fázisban használt mozgási műveletekkel egy-egy új megoldásjelöltet generál az S_i megoldás szomszédságában, majd azokat kiértékeli és kiválasztja közülük a legjobb megoldást (2–11. sorok). Ebben a fázisban csak az inverziós műveletet (6. sor) és a részútvonalterv műveletet (7. sor) használja, mert csak ezek a műveletek garantálják, hogy az új megoldásjelölt megvalósítható lesz. Ezt addig ismétli, amíg az ismert legjobb lokális megoldás $S_i^\star$ nem javítható meghatározott számú iteráción belül (azaz a kora eléri az n_{lsl} értéket). Ha az $S_i^\star$ megoldásjelölt megfelelőségi értéke nagyobb vagy egyenlő, mint az S_i megoldásé, akkor a megoldásjelölt lecseréli az S_i megoldást a populációban (14–15. sorok).

Következő lépésként az algoritmus kiszámítja a p_i nyerési valószínűség értékét minden S_i megoldáshoz (16–19. sorok). A p_i valószínűségi értékek kiszámítása ugyanazzal a függvénnyel történik, mint az alap ABC algoritmusban is (D.1. egyenlet).

Algoritmus 12: *employedBP* (A felfedezésre összpontosító CARP-ABC algoritmus dolgozó méh fázisa)

```
   input  : I; C = {S_1, S_2, ..., S_i, ..., S_{n_cs}}; n_lsl ∈ ℕ
1  begin
2      forall S_i ∈ C do                                   // minden megoldására a kolóniában
3          S_i^⋆ ← S_i;                                    // a szomszédságban talált legjobb megoldás inicializálása
4          α_i^⋆ ← 0;                                      // a szomszédságban talált legjobb megoldás korának inicializálása
5          while α_i^⋆ < n_lsl do
6              S_{i,1} ← inverse(I, S_i);                  // az inverz művelet alkalmazása
7              S_{i,2} ← subRoutePlan(I, S_i);             // a részútvonalterv művelet alkalmazása
8              S'_i ← selectBestSolution({S_i^⋆, S_{i,1}, S_{i,2}});   // a legjobb megoldás kiválasztása
9              if fit(S_i^⋆) < fit(S'_i) then              // ha az iteráció legjobb megoldása jobb mint az eddigi
10                 S_i^⋆ ← S'_i;                           // az eddigi legjobb megoldás lecserélése
11                 α_i^⋆ ← 0;                              // a megoldás korának nullázása
12             else
13                 α_i^⋆ ← α_i^⋆ + 1;                      // az eddigi legjobb megoldás korának növelése
14         if fit(S_i) < fit(S_i^⋆) then                   // ha az eddigi legjobb megoldás jobb, mint a kiinduló megoldás
15             S_i ← S_i^⋆;                                // a kolóniában lévő megoldás lecserélése
16     P ← ∅;                                              // a valószínűségi értékek halmazának inicializálása
17     forall S_i ∈ C do                                   // minden megoldására a kolóniában
18         p_i ← fit(S_i) / Σ_{j=1}^{n_cs} fit(S_j);       // a megoldás valószínűségi értékének kiszámolása
19         P ← P ∪ p_i;                                    // a valószínűségi érték hozzáadása a valószínűségi értékek halmazához
20     return C, P;
```

3.4.5. Megfigyelő méh fázis

A CARP-ABC algoritmus megfigyelő méh fázisának algoritmikus leírása a 13. algoritmusban látható. Az algoritmus ebben a fázisban a p_i értékektől függően kiválaszt egy S_i megoldást a *selectSolutionByProbability* függvénnyel (2. sor). Ez a függvény először $int(\sqrt{n_{cs}})$ alkalommal rulett szelekciót hajt végre, hogy $int(\sqrt{n_{cs}})$ számú megoldást kiválasszon a kolóniából. Ezután összehasonlítja a kiválasztott megoldásokat egymással, és kiválasztja közülük a legjobbat (azaz a legnagyobb megfelelőségi értékkel rendelkező megoldást).

Következő lépésként az algoritmus n_{cs} számú $S_{i,j}$ új megoldásjelöltet generál az S_i megoldások szomszédságában (azaz minden egyes megfigyelő méhre egy megoldást) az egyesítés-szétválás művelet segítségével (5–6. sorok). Ezen megoldások szomszédságában a kis és a közepes lépésméretű műveletekkel új megoldásjelölteket generál, amíg az ismert legjobb helyi megoldás $S^{\star}_{i,j}$ nem javítható a meghatározott számú iteráción belül (azaz a megoldás kora, $\alpha^{\star}_{i,j}$, eléri az n_{lsl} értéket) (7–21. sorok). Ebben a fázisban az összes kis lépésméretű műveletet (azaz az inverziós, a betoldási, a csere és a kétopciós műveleteket) és a közepes lépésméretű részútvonalterv műveletet alkalmazza az $S'_{i,j}$ megoldásra, amely az előző iterációban talált legjobb megoldás (11–15. sorok). Az így kapott megoldások közül a *selectBestSolution* függvénnyel kiválasztja a legjobbat, mint az új $S'_{i,j}$ megoldást (16. sor). Ha az új $S'_{i,j}$ megoldás jobb, mint az $S_{i,j}$ szomszédságában jelenleg ismert legjobb megoldás (azaz $S^{\star}_{i,j}$), akkor a jelenlegi $S^{\star}_{i,j}$ megoldást lecseréli az $S'_{i,j}$ megoldásra (17–19. sorok). Ellenkező esetben az $S^{\star}_{i,j}$ megoldás korát (azaz az $\alpha^{\star}_{i,j}$ értékét) eggyel növeli (20–21. sorok). Miután a keresés véget ér az $S_{i,j}$ megoldás szomszédságában, az algoritmus megvizsgálja, hogy a legjobb talált megoldás (azaz $S^{\star}_{i,j}$) jobb-e, mint az

S_i teljes szomszédságában talált legjobb megoldás (azaz $S_i^\star$). Ha igen, azaz az $S_{i,j}^\star$ megoldás magasabb megfelelőségi értékkel rendelkezik és egyben megvalósítható is (teljes többlet igénye nulla), akkor az a jelenlegi $S_i^\star$ megoldást lecseréli (22–23. sorok).

Ha az ebben a fázisban talált legjobb megoldás (azaz $S_i^\star$) jobb, mint az aktuális S_i megoldás, akkor az $S_i^\star$ megoldás lecseréli azt a kolóniában (25–26. sorok).

Algoritmus 13: *onlookerBP* (A felfedezésre összpontosító CARP-ABC algoritmus megfigyelő méh fázisa)

```
   input : I; C = {S_1, S_2, ..., S_i, ..., S_{n_cs}}; P = {p_1, p_2, ..., p_i, ..., p_{n_cs}}; n_cs, n_lsl ∈ ℕ
1  begin
2      S_i ← selectSolutionByProbability(C, P);          // egy megoldás kiválasztása a kolóniából
3      S_i^⋆ ← S_i;                                      // az S_i szomszédságban talált legjobb megoldás inicializálása
4      j ← 1;                                            // az iteráció számláló inicializálása
5      while j ≤ n_cs do                                 // amíg a megvizsgált megoldások száma nem egyenlő a kolónia méretével
6          S_{i,j} ← mergeSplit(S_i);                    // az egyesítés-szétválasztás művelet alkalmazása S_i-n
7          S_{i,j}^⋆ ← S_{i,j};                          // az S_{i,j} szomszédságában talált eddigi legjobb megoldás inicializálása
8          S'_{i,j} ← S_{i,j};                           // az S_{i,j} szomszédságában talált iterációs legjobb megoldás inicializálása
9          α_{i,j}^⋆ ← 0;                                // az S_{i,j}^⋆ korának inicializálása
10         while α_{i,j}^⋆ < n_lsl do                    // amíg az S_{i,j}^⋆ kora el nem éri a lokális keresési limitet
11             S_{i,j,1} ← inverse(I, S'_{i,j});         // az inverziós művelet alkalmazása S'_{i,j}-n
12             S_{i,j,2} ← insert(I, S'_{i,j});          // a betoldási művelet alkalmazása S'_{i,j}-n
13             S_{i,j,3} ← swap(I, S'_{i,j});            // a csere művelet alkalmazása S'_{i,j}-n
14             S_{i,j,4} ← twoOpt(I, S'_{i,j});          // a kétopciós művelet alkalmazása S'_{i,j}-n
15             S_{i,j,5} ← subRoutePlan(I, S'_{i,j});    // a részútvonalterv művelet alkalmazása S'_{i,j}-n
16             S'_{i,j} ← selectBestSolution({S_{i,j,1}, S_{i,j,2}, S_{i,j,3}, S_{i,j,4}, S_{i,j,5}});  // az iterációs legjobb
                 megoldás kiválasztása
17             if fit(S_{i,j}^⋆) < fit(S'_{i.j}) then    // ha az S'_{i,j} a szomszédság eddigi legjobb megoldása
18                 S_{i,j}^⋆ ← S'_{i,j};                 // a szomszédság eddigi legjobb megoldásának lecserélése
19                 α_{i,j}^⋆ ← 0;                        // a megoldás korának nullázása
20             else
21                 α_{i,j}^⋆ ← α_{i,j}^⋆ + 1;            // a szomszédság eddigi legjobb megoldásának a korának a növelése
22         if fit(S_i^⋆) < fit(S_{i,j}^⋆) ∧ calculateTotalExcessDemand(I, S_{i,j}^⋆) = 0 then  // ha a szomszédság
               eddigi legjobb (megvalósítható) megoldása az S_i szomszédságban talált eddigi legjobb
               megoldás
23             S_i^⋆ ← S_{i,j}^⋆;                        // a megoldás lecserélése
24         j ← j + 1;                                    // az iteráció számláló növelése
25     if fit(S_i) < fit(S_i^⋆) then                     // ha az S_i szomszédságban talált legjobb megoldás jobb, mint S_i
26         S_i ← S_i^⋆;                                  // a kolóniabéli megoldás lecserélése
27     return C;
```

3.4.6. Felderítő méh fázis

A CARP-ABC algoritmus felderítő méh fázisának algoritmikus leírása a 14. algoritmusban látható. Az algoritmus ebben a fázisban növeli a változatlan megoldások korát (3–6. sorok), valamint nullára állítja a kolónián belüli új megoldások korát (sorok 8–9. sorok). Továbbá, ha a kolóniában van egy elhagyott megoldás (azaz egy olyan megoldás, amelyet egy előre meghatározott n_{sal} számú próbálkozással sem sikerült javítani), az algoritmus egy új, RSG algoritmus által generált megoldásra cseréli le (5–7. sorok).

Az algoritmus ebben a fázisban frissíti az $S^\star$ globálisan legjobb megoldást. Először az új kolónia legjobb megoldását kiválasztja a *selectBestSolution* függvénnyel S' megoldásként (10. sor). Ha az S' megoldás jobb, mint az $S^\star$ megoldás, akkor az lesz az új globálisan legjobb megoldás (11–13. sorok). Ellenkező esetben az $S^\star$ megoldás korát (azaz az $\alpha^\star$ értékét) eggyel növeli (14–15. sorok).

Algoritmus 14: *scoutBP* (A felfedezésre összpontosító CARP-ABC algoritmus felderítő méh fázisa)

```
   input  : I; C̄ = {S̄_1, S̄_2, ..., S̄_i, ..., S̄_{n_cs}}; C = {S_1, S_2, ..., S_i, ..., S_{n_cs}};
            A = {α_1, α_2, ..., α_i, ..., α_{n_cs}}; S*; α*, n_sal ∈ ℕ
1  begin
2      forall S_i ∈ C do                                   // minden megoldásra a kolóniában
3          if fit(S_i) = fit(S̄_i) then                     // ha a megoldás (megfelelősége) változatlan
4              α_i ← α_i + 1;                              // a megoldás korának növelése
5              if α_i = n_sal then                         // ha a megoldás kora elérte a korhatárt
6                  S_i ← RSG(I);                           // új megoldás létrehozása az RSG algoritmussal
7                  α_i ← 0;                                // a megoldás korának inicializálása
8          else
9              α_i ← 0;                                    // a megoldás korának nullázása
10     S' ← selectBestSolution(C);                         // a legjobb megoldás kiválasztása a kolóniából
11     if fit(S*) < fit(S') then // ha a megoldás jobb, mint a jelenleg ismert globálisan legjobb megoldás
12         S* ← S';                                        // a megoldás lecserélése
13         α* ← 0;                                         // az S* megoldás korának nullázása
14     else
15         α* ← α* + 1;                                    // az S* megoldás korának növelése
16     return C, A, S*, α*;
```

3.4.7. Módosítások

Ebben az alszakaszban a **CARP-ABC algoritmus feltárásra összpontosító változata** kerül bemutatásra, a felfedezésre összpontosító változathoz viszonyítva.

A fő algoritmus

A felfedezésre összpontosító CARP-ABC algoritmushoz képest ennek az algoritmusnak több paramétere van. Ezek a következők:

- n_{nil}: nincs javulási korlát a megfigyelő méh fázisban;
- S: kezdeti megoldás, ami a kezdeti kolónia egyik tagja lesz (megadása opcionális);
- *time_limit*: időkorlát a teljes algoritmus futásához.

Az algoritmus a futása közben több ponton is ellenőrzi, hogy az eltelt idő elérte-e vagy meghaladta-e a *time_limit* értékét. Ha igen, akkor az algoritmus működése leáll és visszaadja az addig talált globálisan legjobb megoldást.

Mivel időkorlát használata esetén az algoritmus működése bármely fázisban leállhat, az addig ismert globálisan legjobb megoldást nem csak a felderítő méh fázisban frissítheti, hanem a többi fázisban is. Ha bármely fázisban végrehajtott keresés folyamán az algoritmus egy olyan megoldást talál, ami jobb mint a szomszédságban talált addigi legjobb megoldás, akkor azt is megnézi, hogy az adott megoldás az addig ismert globálisan legjobb megoldásnál is jobb-e. Ha igen, akkor az addig ismert globálisan legjobb megoldást lecseréli az adott megoldásra.

További különbség, hogy a mozgási műveletek két feladatot (vagy feladatazonosítót) is fogadnak bemenetként (pl. 15. algoritmus 18. sora), nem csak a példát és a megoldást. Ezáltal nem csak véletlenszerűen választott feladatokkal lehet mozgási műveleteket végrehajtani.

Inicializálási fázis

Az algoritmus a véletlenszerű megoldás generátor (RSG) algoritmus helyett a véletlenszerű útkeresési heurisztikát (RPSH) használja a kezdeti populáció létrehozásához. Így finomítottabb kezdeti megoldásokkal tud dolgozni.

Dolgozó méh fázis

Az algoritmus a keresést csak a populáció új tagjaira végzi el. Emellett, a jobb szomszédos megoldások kereséséhez (az inverziós és a részútvonalterv műveletek helyett) a nagy lépésméretű egyesítés-szétválasztás műveletet használja. Erre azért van szükség, mivel a felderítő méh fázisban az RSG algoritmust használja az új megoldás(ok) létrehozásához, tehát az(ok)on nagyobb finomítások szükségesek. Ha a finomítást követően kapott megoldás jobb, mint a szomszédságban talált eddigi legjobb megoldás ($S_i^\star$), akkor azt is megvizsgálja, hogy az az eddig ismert globálisan legjobb megoldásnál ($S^\star$) is jobb-e. Ha igen, akkor azt lecseréli. A keresés végén – hasonlóan a felfedezésre összpontosító CARP-ABC algoritmushoz – a populáció tagját lecseréli a szomszédságban talált legjobb megoldásra. Viszont ebben a változatban a keresés befejezését követően nem számol valószínűségi értékeket.

Megfigyelő méh fázis

A megfigyelő méh fázis működését a 15. algoritmus írja le részletesen. Az algoritmus a fázis legelején a lokális kereséshez a populációból kiválasztja a legjobb S_i megoldást (tehát itt nem valószínűségi értékek alapján történik a választás) (2. sor).

Algoritmus 15: *onlookerBPv2* (A feltárásra összpontosító CARP-ABC algoritmus megfigyelő méh fázisa)

```
   input  : I; C = {S_1, S_2, ..., S_i, ..., S_{n_cs}}; P = {p_1, p_2, ..., p_i, ..., p_{n_cs}}; n_cs, n_lsl ∈ ℕ
1  begin
2      S_i ← selectBestSolution(C);                          // a legjobb megoldás kiválasztása a kolóniából
3      O ← ⟨twoOpt, swap, insert, subRoutePlan⟩;             // a használt műveletek definiálása
4      S_i^⋆ ← S_i;                                          // az S_i szomszédságában talált eddigi legjobb megoldás inicializálása
5      α_i^⋆ ← 0;                                            // az S_i^⋆ korának inicializálása
6      while α_i^⋆ ≤ n_nil do                                // amíg az az S_i^⋆ kora el nem éri a nincs javulási korlátot
7          imp ← false;                                      // (egy iterációra vonatkozó) javulást jelző logikai érték inicializálása
8          O ← shuffle(O);                                   // a mozgási műveletek sorrendjének megkeverése
9          forall o ∈ O do                                   // minden mozgási műveletre
10             T_i ← getTasks(S_i^⋆);                        // a (nem ál)feladatok kinyerése a megoldásból
11             T_i ← shuffle(T_i);                           // a feladatok sorrendjének megkeverése
12             forall t_1 ∈ T_i do                           // az S_i^⋆ megoldás minden (nem ál)feladatára
13                 L_{t_1} ← T_i;                            // a jelölt lista inicializálása
14                 if n_nil/2 > α_i^⋆ then
15                     L_{t_1} ← createCandidateList(I, t_1, T_i);   // a jelölt lista létrehozása
16                 L_{t_1} ← shuffle(L_{t_1});               // a feladatok sorrendjének megkeverése
17                 forall t_2 ∈ L_{t_1} do                   // a jelölt lista minden feladatára
18                     S'_i ← o(I, S_i^⋆, t_1, t_2);         // a mozgási művelet alkalmazása S_i^⋆-ra
19                     if fit(S_i^⋆) < fit(S'_i) then        // ha S'_i a szomszédság eddigi legjobb megoldása
20                         S_i^⋆ ← S'_i;                     // a megoldás lecserélése
21                         α_i^⋆ ← 0;                        // az S_i^⋆ megoldás korának nullázása
22                         imp ← true;                       // a javulást jelző érték frissítése
23                         if fit(S^⋆) < fit(S'_i) then      // ha S'_i az eddig ismert globálisan legjobb
                               megoldás
24                             S^⋆ ← S';                     // a megoldás lecserélése
25                             α^⋆ ← 0;                      // az S^⋆ megoldás korának nullázása
26             if imp = false then                           // ha (ebben az iterációban) nem volt javulás
27                 α_i^⋆ ← α_i^⋆ + 1;                        // az S_i^⋆ megoldás korának növelése
28     if fit(S_i) < fit(S_i^⋆) then  // ha az S_i szomszédságban talált eddigi legjobb megoldás jobb mint S_i
29         S_i ← S_i^⋆;                                      // a kolóniabéli megoldás lecserélése
```

A keresés folyamán mindig az aktuálisan legjobbnak vélt megoldáson ($S_i^\star$) végzi a műveleteket (18. sor). Ha ennek a megoldásnak az aktuális kora kisebb mint a nincs javulási korlát fele (azaz $n_{nil}/2 > \alpha_i^\star$, 14. sor), akkor a jelölt lista (lásd következő bekezdés) szerint végzi a keresést (azaz $n_{nil}/2 > \alpha_i^\star$, 15. sor). Ha több iteráción

keresztül sem tud a jelölt listával jobb megoldást találni (azaz a megoldás kora meghaladja a nincs javulási korlát felét, $n_{nil}/2 < \alpha_i^\star$), akkor véletlenszerűen választott mozgási műveletekkel próbálkozik. Ha sikerül a szomszédságban jobb megoldást találnia (19. sor), akkor $S_i^\star$ megoldást lecseréli és így a kora ($\alpha_i^\star$) is nullázódik (20–21. sorok). Ha nem sikerül jobb megoldást találnia és az $\alpha_i^\star$ eléri n_{nil} értékét (6. sor), akkor a keresést befejezi és az S_i megoldást lecseréli a szomszédságban talált legjobb megoldásra (28–29. sorok).

A jelölt lista (L_{t_1}) tartalma a még kiszolgálandó feladatok azon halmaza, amelyek a legközelebb vannak (az *mdc* függvény szerint) az éppen vizsgált $t_1 \in T_i$ feladathoz (12. sor). Tehát a $t_2 \in L_t$ feladatok (17. sor) az alapján vannak növekvő sorrendbe rendezve, hogy milyen közel vannak a $t_1 \in T_i$ feladathoz.

A felfedezésre összpontosító CARP-ABC algoritmussal ellentétben nem használja az inverziós műveletet, mivel a feladatok inverzét a többi mozgási művelet is figyelembe veszi. Emellett az is megemlítendő, hogy valós úthálózatok esetében nem gyakoriak a kétirányú forgalmat megengedő útszakaszok.

Felderítő méh fázis

Az algoritmus – hasonlóan a felfedezésre összpontosító CARP-ABC algoritmushoz – az RSG algoritmust használja az új megoldás(ok) létrehozásához, hogy valamelyest növelje a vizsgált megoldás(ok) változatosságát. Az új megoldáso(ka)t a dolgozó méh fázisban finomítja.

3.5. A javasolt adat-vezérelt DCARP keretrendszer és az RR1 újratervező algoritmus

Ebben a szakaszban a javasolt adat-vezérelt DCARP keretrendszerek és az általuk alkalmazott RR1 újratervező algoritmus kerül bemutatásra. Először a javasolt megközelítés egészének áttekintése, majd az RR1 újratervező algoritmus részletesebb leírása következik.

3.5.1. Áttekintés

A javasolt adat-vezérelt DCARP keretrendszerből két változat készült:

- egy esemény generátort használó DCARP keretrendszer (3.8. ábra), és
- egy forgalom szimulációt használó DCARP keretrendszer (3.9. ábra).

Mindkét változat célja, hogy valósághű szimulációs környezetet biztosítson a (D)CARP megoldók teszteléséhez. A kettő közötti fő különbség a szimulációhoz használt adatok összetételében és minőségében van. A használt modulok működése nagymértékben megegyezik; csak az esemény létrehozó modul esetében vannak eltérések. A modulok a következők:

- Az **inicializáló modul** a kezdeti (CARP) megoldó használatával inicializálja a problémát és létrehozza a kezdeti szolgáltatási tervet. Bármely statikus CARP megoldót képes használni, de alapértelmezettként a HMA-t alkalmazza, mivel jelenleg az képes a legjobb megoldásokat nyújtani [36].

- Az **esemény létrehozó modul** gondoskodik a szimulációhoz szükséges események létrehozásáról. Az úthálózathoz és a járműhöz kapcsolódó eseményeket alapból véletlenszerűen generálja, de ez szabályozható. A hatékonyság vizsgálatok során a lehetséges váratlan események közül csak a kritikus események (azaz a „feladat megjelenés", az „igény növekedés" és a „jármű meghibásodás" események) lettek generálva, mivel csak azok bekövetkezését követően válhat a szolgáltatási terv olyan mértékben megvalósíthatatlanná, hogy azt egyszerű módosítással nem lehet korrigálni, a szolgáltatási terv újratervezésére (azaz DCARP megoldó használatára) van szükség.

- Az **eseménykezelő modul** minden lehetséges eseményt képes kezelni. Az ábrákon (3.8. ábra és 3.9. ábra) feltüntetett események esetében külön kezelő algoritmusok vannak használva, mivel azoknál a példa (és a szolgáltatási terv) összetettebb módosítást igényelhet. A nem feltüntetett események esetében elegendő a megfigyelt értékek alapján frissíteni a példát. A vastagon szedett események kritikus (és félkritikus) események, amik esetében újratervezés lehet szükséges.

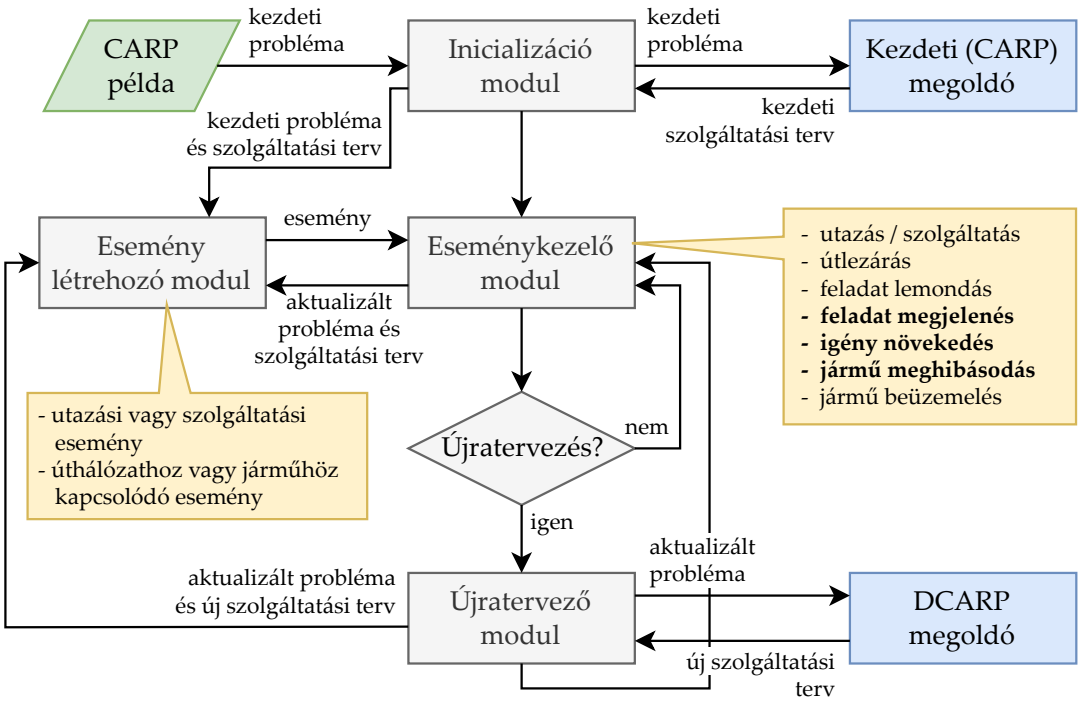
- Az **újratervező modul** bármely statikus CARP megoldót képes DCARP megoldóként használni. Alapértelmezettként az ebben a munkában bemutatott RR1 újratervező algoritmust, majd a CARP-ABC algoritmust alkalmazza, de a hatékonyság vizsgálatok folyamán más algoritmusok, illetve algoritmus kombinációk is alkalmazásra kerültek. (A vizsgálatokhoz felhasznált algoritmusok a 3.6.1. alszakaszban kerülnek bemutatásra.)

Mindkét változat esetén az algoritmus a futását az **inicializáló modul**ban kezdi. Ezt követően az **eseménykezelő modul** aktiválódik, hogy figyelje az **esemény létrehozó modul** (és a forgalom szimulációs szoftver) által biztosított eseményeket. Ha olyan kritikus (vagy félkritikus) esemény következik be, ami módosítja az aktuális szolgáltatási terv megvalósíthatóságát (és/vagy teljes költségét nagy mértékben), akkor az **újratervező modul** aktiválódik. A használt DCARP megoldó, megkapva az aktualizált problémát és egy lehetséges megoldási lehetőséget, megpróbál egy jobb minőségű megoldást találni. Az aktualizált problémát és az új szolgáltatási tervet az újratervező modul továbbítja az eseménykezelő modul és az esemény létrehozó modul számára. A forgalom szimulációs szoftvert használó keretrendszer esetében az új szolgáltatási tervet a forgalom szimulációs szoftver is megkapja.

Az esemény generátort használó adat-vezérelt DCARP keretrendszer működése a D. függelék D.5. szakaszában kerül részletes bemutatásra. Az újratervező modul részét képező RR1 újratervező algoritmus leírása a 3.5.2. alszakaszban található.

Az esemény generátort használó adat-vezérelt DCARP keretrendszer

Az esemény generátort használó adat-vezérelt DCARP keretrendszer (3.8. ábra) egy valós feladaton alapuló CARP példát használ a szimulációhoz szükséges (esemény)adatok előállításához. Az **esemény létrehozó modul** biztosítja a szolgáltatási folyamat végrehajtását leíró eseményeket (azaz az utazási és a szolgáltatási eseményeket), valamint a váratlanul bekövetkező, példát módosító eseményeket (azaz az úthálózathoz és a járműhöz kapcsolódó eseményeket) is. Az utazási és a szolgáltatási események generálása alapértelmezettként úgy történik, hogy a járművek követik a hozzájuk rendelt útvonaltervet, de ez módosítható.



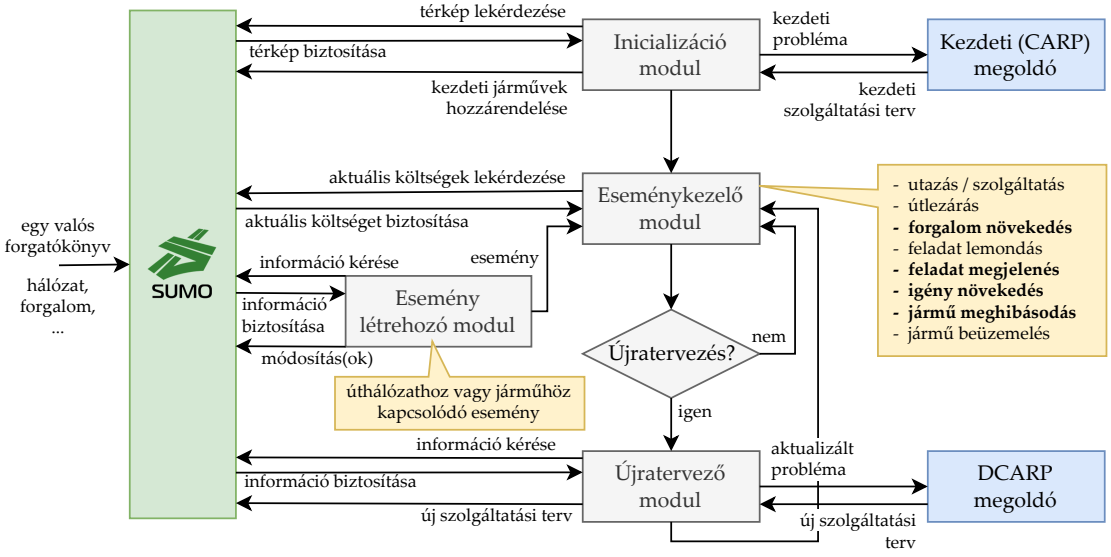
3.8. ábra. Az esemény generátort használó adat-vezérelt DCARP keretrendszer áttekintése

A forgalom szimulációt használó adat-vezérelt DCARP keretrendszer

A forgalom szimulációt használó adat-vezérelt DCARP keretrendszer (3.9. ábra) már egy valósághűbb tesztelési környezetet biztosít, mivel valós úthálózatot és forgalmi adatokat használ a szimulációhoz. A keretrendszer egy folytonos forgalomszimulációs szoftverhez (*Simulation of Urban MObility*, röviden *SUMO* [146]) van csatlakoztatva, ami a forgalmi viszonyokat a megadott úthálózat és forgalmi követelmények szerint szimulálja. Ha mindkettő a valós életből származik, akkor így egy valósághű közúti környezetet lehet létrehozni. A SUMO egy olyan interfészt biztosít, amely lehetővé teszi az útszakaszok (ívek) aktuális áthaladási költségének, valamint a bejárt útvonalnak és a járművek aktuális pozíciójának lekérdezését. Emellett lehetőség van megváltoztatni a járművek útvonaltervét, amit a szolgáltatási terv megváltozása esetén szükséges elvégezni.

Az esemény generátort használó adat-vezérelt DCARP keretrendszerrel ellentétben nincs szükség utazási és szolgáltatási események generálására, mivel azt a forgalom szimulációs szoftver elvégzi. Az úthálózathoz és a járműhöz kapcsolódó események közül a „forgalom csökkenés” és a „forgalom növekedés” események azok, amikről szintén a forgalom szimulációs szoftver gondoskodik. A többi esemény létrehozásához azonban továbbra is szükség van az esemény létrehozó modul használatára. További különbség, hogy az eseménykezelő modul a „forgalom növekedés” események bekövetkezése esetén is meghívhatja az újratervező modult, ha a teljes költség huzamosabb időre nagy mértékben megnő. Az, hogy mi az a mérték ami már túl nagynak számít, paraméterrel szabályozható. Ennek az alapértlemezett értéke 10%.

Ez a keretrendszer egy másik, irodalomban fellelhető DCARP keretrendszeren alapul ([147]). A fő különbség a kettő között, hogy a másik DCARP keretrendszer nem használ esemény létrehozó modult és csak a költségváltozást figyeli. A hozzáadásra kerülő feladatok előre vannak definiálva (nem véletlenszerűen vannak kiválasztva) és az újratervező modul adja azokat hozzá a problémához.



3.9. ábra. A forgalom szimulációt használó adat-vezérelt DCARP keretrendszer áttekintése

3.5.2. Az RR1 újratervező algoritmus

Ebben az alszakaszban az RR1 újratervező algoritmus (16. algoritmus) kerül bemutatásra. Az algoritmus bemenetei az aktuális I DCARP példa, az S megoldásjelölt, az $\overline{S}$ csonka megoldásjelölt, valamint az $\overline{S}$ csonka megoldásba beszúrandó feladato(ka)t tartalmazó T_k feladatsorozat. Az S megoldás egy külön útvonaltervben tartalmazza a T_k feladatsorozat összes feladatát, az $\overline{S}$ csonka megoldás pedig ugyanaz, mint az S megoldás, csak a T_k sorozatban szereplő feladat(ok) nélkül (így eggyel kevesebb útvonaltervvel rendelkezik). Az algoritmus célja a T_k sorozatban szereplő feladat(ok) beillesztése az $\overline{S}$-be a legköltséghatékonyabb módon. Az eddigi legjobb megoldást mindig az S' változóba menti, amit a futása befejeztével átad az újratervező modulnak.

Az algoritmus először kiszámítja a T_k feladatsorozatban lévő feladat(ok) teljes igényét (dem_{T_k}) (2–4. sor), majd az *ids* halmazba összegyűjti azon útvonaltervek azonosítóit, amelyek elegendő maradék kapacitással rendelkeznek a T_k-ban lévő összes feladat kiszolgálásához (5–8. sor). Egy jármű útvonaltervére vonatkozó terhelést (azaz a kiszolgált és a kiszolgálandó feladatok igényeinek összegét) a *load* terhelési függvénnyel számítja ki az $\overline{S}$ csonka megoldás minden egyes $r_i \in \overline{S}$ útvonaltervére. Feltételezzük, hogy a jármű üresen indul begyűjtési szolgáltatás (pl. városi hulladékgyűjtés) esetén, és feltöltve szétosztási szolgáltatás (pl. az utcák sózása) esetén. Ha nincs elegendő kapacitással rendelkező útvonalterv (azaz az *ids* halmaz üres halmaz marad), akkor a feladato(ka)t csak egy új útvonaltervben lehet kiszolgálni. Ilyen esetben az S megoldásjelöltet adja vissza új S' megoldásként (9. sor). A feladat(ok)

biztosan beilleszthetők egy új útvonaltervbe, mivel feltételezzük, hogy egy esemény keretében egyszerre csak egy jármű hibásodhat meg és a járművek homogének (az-az azonos maximális kapacitással rendelkeznek). Ha van legalább egy útvonalterv amibe be lehet illeszteni a feladato(ka)t, akkor minden $i \in ids$ azonosítójú útvonaltervbe megpróbálja egyenként beilleszteni az(oka)t abba a pozícióba, amely a legkisebb többletköltséget generálja (11–39. sor).

Algoritmus 16: $RR1$ (Újratervező algoritmus)

```
   input  : I = (V, v_0, A, T, T_v, n, w, q, H, H_e, H_f, R, R_e, rt, rv, head, tail, dc, inv, dem, sc, mdc),
            S = {r_1, r_2, ..., r_K, r_{K+1}}, S̄ = {r_1, r_2, ..., r_K}, T_k ∈ T*
 1  begin
 2      dem_{T_k} ← 0;                                   // a T_k feladat(ok) teljes igényének a kiszámítása
 3      forall a ∈ T_k do
 4          dem_{T_k} ← dem_{T_k} + dem(a);
 5      ids ← ∅;          // a T_k feladato(ka)t kiszolgálni képes útvonaltervek azonosítójának összegyűjtése
 6      forall i ∈ {1, 2, ..., K} \ R_e do
 7          if load(I, r_i) + dem_{T_k} ≤ q then
 8              ids ← ids ∪ {i};
 9      S' ← S;                                          // a legjobb megoldást tároló változó
10      if ids ≠ ∅ then
11          tc' ← TC(I, S');
12          forall i ∈ ids do
13              S_i ← S̄;                  // az r_i-re vonatkozóan a legjobb megoldást tároló változó
14              tc_i ← TC(I, S̄);
15              forall a ∈ T_k do
16                  Δtc_i ← ∞;              // a legjobb teljes költség különbséget tároló változó
17                  j_a ← −1;                                // a legjobb pozíciót tároló változó
18                  inv_a ← false;
19                  forall t_{i,j} ∈ r_i s.t. t_{i,j} ∉ {v_0} ∪ T_k do          // beszúrási pozíciók viszgálata
20                      Δtc_{i1} ← sc(a) + mdc(tail(t_{i,j−1}), head(a)) + mdc(tail(a), head(t_{i,j})) −
                          mdc(tail(t_{i,j−1}), head(t_{i,j}));          // költségváltozás a beszúrása esetén
21                      Δtc_{i2} ← ∞;
22                      if a ∈ dom(inv) then
23                          Δtc_{i2} ← sc(inv(a)) + mdc(tail(t_{i,j−1}), head(inv(a))) +
                              mdc(tail(inv(a)), head(t_{i,j})) − mdc(tail(t_{i,j−1}), head(t_{i,j}));
                              // költségváltozás inv(a) beszúrása esetén
24                      if Δtc_{i1} < Δtc_i then          // ha az a eddigi legjobb pozíciója j az r_i-ben
25                          Δtc_i ← Δtc_{i1};
26                          j_a ← j;
27                          inv_a ← false;
28                      if Δtc_{i2} < Δtc_i then      // ha az inv(a) eddigi legjobb pozíciója j az r_i-ben
29                          Δtc_i ← Δtc_{i2};
30                          j_a ← j;
31                          inv_a ← true;
32                  t ← a;
33                  if inv_a = true then
34                      t ← inv(a);
35                  S_i ← insertTask(S_i, i, j_a, t);                    // a feladat beszúrása
36                  tc_i ← tc_i + Δtc_i;                          // a teljes költség frissítése
37              if tc_i ≤ tc' then          // ha r_i az eddigi legjobb útvonalterv a feladat(ok) beszúrásához
38                  tc' ← tc_i;
39                  S' ← S_i;
40      return S';
```

Az algoritmus inicializálja az eddigi legjobb megoldást (S') és annak teljes költségét (tc') tároló változókat az S megoldással (9. sor) és annak teljes költségével (11. sor), majd megkezdi a keresési folyamatot. Az ids halmazban szereplő minden egyes i azonosítójú útvonalterv esetében a legjobb megoldás ideiglenesen az S_i változóban kerül eltárolásra, annak teljes költsége pedig a tc_i változóban. Az algoritmus minden egyes iterációs lépésében ezeket a változókat az $\overline{S}$ megoldással és annak teljes költségével inicializálja (13–14. sor), majd megkezdi az iterációt a T_k feladatsorozat elemeiben, amely során minden egyes $a \in T_k$ feladathoz megkeresi a legjobb pozíciót az $r_i \in S_i$ útvonaltervben (15–39. sor). Ehhez a keresési folyamathoz ideiglenesen

eltárolja a feladat beszúrásából eredő legkisebb teljes költségváltozást (Δtc_i), a beszúrás pozícióját (j_a), és azt, hogy a teljes költség szempontjából milyen irányból érdemes kiszolgálni a feladatot ($inv(a)$, ami azt jelzi, hogy az $inv(a)$ jobb-e, mint az a). Ezeket a változókat rendre a végtelen (∞), -1 és $false$ értékekkel inicializálja (16–18. sor).

Az algoritmus ezután megkezdi az iterációt az r_i útvonalterven belüli lehetséges j pozíciók felett (19–31. sor). Minden $a \in T_k$ feladat csak az álfeladatok (azaz a t_0 feladatok) közé és (ha van) csak a virtuális feladat után illeszthető be. Minden egyes j pozícióra kiszámítja Δtc_{i1} értékét, ami az a feladatnak az r_i útvonalterven belüli j pozícióba történő beszúrásából származó teljes költségváltozást jelenti (20. sor). Ha az a feladat egy élfeladat (azaz $a \in dom(inv)$), akkor az $inv(a)$ beszúrásából származó teljes költségváltozást (azaz Δtc_{i2} értékét) is kiszámítja (21–23. sor). Ha a j az eddigi legjobb pozíció az r_i útvonaltervben az a feladat beszúrásához (azaz j-vel kisebb teljes költségváltozás valósítható meg, mint j_a-val), akkor az lesz az új legjobb pozíció (azaz j_a új értéke) (24–27. sor). Ugyanez vonatkozik az $inv(a)$-ra is (28–31. sor). Miután az összes pozíciót megvizsgálta, az a feladatot (vagy az $inv(a)$ feladatot, ha $inv_a = true$) az $insert$ függvénnyel beszúrja az S_i megoldás r_i útvonaltervén belül a j_a pozícióba (32–35. sor). Az S_i megoldás tc_i teljes költségét is növeli a legkisebb ismert Δtc_i teljes költségváltozással (azaz a feladat beillesztése utáni teljes költségváltozással) (36. sor).

Miután a T_k feladatsorozatban lévő összes feladatot beillesztette az S_i megoldásba, az algoritmus ellenőrzi, hogy ez a megoldás jobb-e (azaz kisebb-e a teljes költsége), mint a jelenleg ismert legjobb S' megoldás (37. sor). Ha az S_i megoldás jobb, akkor az lesz az új legjobb megoldás, és a teljes költsége lesz az új legjobb teljes költség (38–39. sor).

3.6. A javasolt megoldások hatékonyságának vizsgálata

Az ebben a fejezetben bemutatott megoldásokhoz kapcsolódóan a következő vizsgálatok lettek elvégezve:

- **Művelet hatékonyság vizsgálat** (3.6.2. alszakasz): A vizsgálat célja a részútvonalterv művelet hatékonyságának a mérése az irodalomban fellelhető (CARP-hoz használt) mozgási műveletekéhez viszonyítva. A vizsgálat keretében a műveletek hatékonysága különböző bonyolultsági szintű CARP példákon lettek mérve.

- **Hatékonyság vizsgálat CARP példákon** (3.6.3. alszakasz): A vizsgálat célja a felfedezésre összpontosító CARP-ABC algoritmus hatékonyságának mérése, mint CARP megoldó algoritmus, az irodalomban fellelhető CARP megoldó algoritmusokhoz viszonyítva. A vizsgálat keretében az algoritmusok hatékonysága különböző bonyolultsági szintű CARP példákon lettek mérve.

- **Hatékonyság vizsgálat DCARP példákon az esemény generátort használó DCARP keretrendszer használatával** (3.6.4. alszakasz): A vizsgálat célja a feltárásra összpontosító CARP-ABC algoritmus és az RR1 algoritmus hatékonyságának mérése, mint DCARP megoldó algoritmus, az irodalomban

fellelhető (D)CARP megoldó algoritmusokhoz viszonyítva. Ehhez az ebben a munkámban bemutatott esemény generátort használó DCARP keretrendszer lett felhasználva. A vizsgálat keretében az algoritmusok hatékonysága egy CARP példából generált kritikus esemény bekövetkezésére lett mérve (mindhárom kritikus eseményre külön-külön).

- **Hatékonyság vizsgálatok DCARP példákon a forgalom szimulációt használó DCARP keretrendszer használatával** (3.6.5. alszakasz): A vizsgálatok célja a feltárásra összpontosító CARP-ABC algoritmus hatékonyságának mérése (RR1 algoritmussal támogatva és anélkül), mint DCARP megoldó algoritmus, az irodalomban fellelhető (D)CARP megoldó algoritmusokhoz viszonyítva. Ehhez az ebben a munkámban bemutatott forgalom szimulációt használó DCARP keretrendszer lett felhasználva. A vizsgálatok keretében az algoritmusok hatékonysága több forgatókönyvön és többfajta esemény bekövetkezésének a sorozatán lett mérve.

A vizsgálatokhoz felhasznált (D)CARP megoldó algoritmusok és azok használatához szükséges paraméterbeállítások a 3.6.1. alszakaszban kerülnek ismertetésre.

A vizsgálatokhoz felhasznált példák a D. függelék D.6. alszakaszában kerülnek bemutatásra. Az első három pontban feltűntetett vizsgálatokhoz a szakirodalomban gyakran használt teljesítménymérő tesztkészletekből kerültek példák felhasználásra. Az utolsó pontban megemlített vizsgálathoz pedig egy város forgalmi adatainak felhasználásával generált DCARP forgatókönyvek kezdő példái lettek alkalmazva.

A vizsgálatok elvégzéséhez a DCARP keretrendszer mindkét verziója, a javasolt CARP-ABC algoritmusok (a részútvonalterv művelettel együtt), valamint az irodalomban felelhető további (D)CARP megoldó algoritmusok is Python (3.6) nyelven lettek implementálva. Az első három vizsgálathoz kapcsolódó implementációk és a vizsgálatokat végrehajtó kódok elérhetők a GitHub-on[1]. A forgalom szimulációt használó DCARP keretrendszer implementációja és az ahhoz kapcsolódó vizsgálatokat végrehajtó kódok egy másik repository-ban érhetők el a GitHub-on[2]. A vizsgálatokat Windows 10 operációs rendszerű, Intel(R) Core(TM) i5-3320M 2,60 GHz-es, kétmagos processzorral és 8 GB RAM-mal felszerelt laptopon végeztem.

3.6.1. A felhasznált (D)CARP megoldó algoritmusok

A DCARP megoldó algoritmusok két fő kategóriába sorolhatók:

- **Determinisztikus algoritmusok**, amelyek kifejezetten csak DCARP-ok megoldására lettek kifejlesztve. Jellemzően lokális keresést végeznek egyetlen egy megoldáson, amihez szükséges egy kezdeti megoldást megadni. Ide tartozik az ebben a munkában bemutatott RR1 algoritmus. Ide tarozik még a hibrid lokális kereső (*Hybrid Local Search*, röviden *HyLS*) algoritmus is, ami addig végez lokális keresést, ameddig jobb megoldásokat már nem talál.

- **Nem determinisztikus algoritmusok**, amelyek a statikus CARP megoldók módosított változatai. Általában egy megoldás halmazból választott megoldáson hajtanak végre alaposabb keresést. A használatukhoz nem szükséges, de

[1] `https://github.com/zsuzsanna-nagy/carp-abc`

[2] `https://github.com/zsuzsanna-nagy/dcarp_framework_with_sumo`

van lehetőség kezdeti megoldás(oka)t megadni. A megoldáshalmaz többi elemét saját maguk hozzák létre. Ide tartozik az ebben a munkában bemutatott CARP-ABC algoritmusok, valamint az irodalomban fellelhető HMA is.

A hatékonyság vizsgálatokhoz felhasznált irodalomban fellelhető (D)CARP megoldó algoritmusokat (HMA, ACOPR és HyLS) a vonatkozó publikációkban olvasható leírásuk alapján implementáltam ([36], [37] és [148]), mivel az eredeti implementált változatuk nem állt rendelkezésemre. Ebből adódóan, az ebben a munkában használt implementációk tartalmazhatnak olyan hibákat, amelyek (nem szándékosan) csökkentik ezen algoritmusok hatékonyságát. Az algoritmusokat csak a vonatkozó munkákban meghatározott optimális paraméter beállításokkal használtam.

Az esemény generátort használó DCARP keretrendszer használatával végzett hatékonyság vizsgálat során a HMA esetében az új globálisan legjobb megoldás csak az algoritmus aktuális ciklusának a végén került rögzítésre, mivel ez az eredeti implementációban is így van. A forgalom szimulációt használó DCARP keretrendszer használatával végzett hatékonyság vizsgálatokhoz ezen egy kicsit változtattam. A módosítás lehetővé teszi, hogy az új globálisan legjobb megoldás a megtalálását követően minél előbb rögzítésre kerüljön. Ez növeli a HMA-n belüli vizsgálatok számát, így valamelyest a futási idejét is, de ez elhanyagolható.

A részútvonalterv mozgási művelet a GSTM műveleten [143] alapszik, ezért a részútvonalterv művelet paraméterértékeiként ugyanazokat használtam, mint ami a GSTM művelet számára is optimális. Ezek a következők: $p_{rc} = 0,5$, $p_{cp} = 0,8$, $p_l = 0,2$, $l_{\min} = 2$, $l_{\max} = Int(\sqrt{n})$, és $n_{\max} = 5$, ahol n a kiválasztott útvonalterven belüli feladatok számát jelöli.

A felfedezésre összpontosító CARP-ABC algoritmus esetében a következő paraméterértékek adták a legjobb minőségű eredményeket anélkül, hogy az algoritmus futási ideje túlzottan megnövekedett volna: $n_{cs} = 1$, $n_{mi} = 10.000$, $n_{gsl} = 100$, $n_{lsl} = 20$, és $n_{sal} = 20$. A feltárásra összpontosító CARP-ABC algoritmus esetében pedig a következő paraméterértékek: n_{cs}= 10, $n_{lsl} = 10$, $n_{nil} = 5$. A vizsgálatok során ezeket a paraméterbeállításokat alkalmaztam.

3.6.2. A részútvonalterv művelet hatékonyságának vizsgálata

Vizsgálati környezet és paraméterek

A művelet hatékonyság vizsgálathoz a D. függelék D.6. szakaszában bemutatott három különböző bonyolultsági szintű CARP példából használtam fel egyet-egyet: a közepesen nehéz **egl-e1-A**, a nehéz **egl-s1-A** és a legnehezebb **egl-g1-A** példákat. A részútvonalterv művelet és a CARP megoldásához használt kis lépésméretű műveletek (inverziós, betoldási, csere és kétopciós műveletek) a felfedezésre összpontosító ABC algoritmus dolgozó méh fázisán belüli lokális kereséshez használt műveletként kerültek alkalmazásra. Azért a dolgozó méh fázis lett választva a megfigyelő méh fázis helyett, hogy a műveletek hatékonyságát különböző minőségű megoldásokon is meg lehessen mérni, ne csak a jó minőségű megoldásokon.

A (módosított) felfedezésre összpontosító ABC algoritmust minden példára 30-szor és 10 perces futási idő korlátozással futtattam le, egymástól függetlenül. Az algoritmus végrehajtása során rögzítésre került azon keresési próbálkozások száma, amelyek során a mozgási művelet lokálisan jobb megoldást (azaz $S_i^\star$-t) talált.

Eredmények

A művelet hatékonyság vizsgálat eredményeit a 3.3. táblázat foglalja össze. Minden cellában az látható százalékos arányként, hogy az oszlop fejlécében megnevezett művelet az első oszlopban megadott példa esetében az összes olyan lefutás hány százalékában talált jobb megoldást, ahol valamelyik művelet jobb megoldást talált. Az egyszerűség kedvéért hívjuk ezt a mérőszámot hatékonyságnak. Látható, hogy mindhárom példa esetében az általam javasolt részútvonalterv művelet hatékonysága a legmagasabb, tehát a vizsgált műveletek közül – a probléma méretétől függetlenül – a legnagyobb eséllyel ez a művelet javítja az aktuális megoldást a keresési folyamat során.

3.3. táblázat. A műveletek hatékonyságának összehasonlítása

Példa	**Művelet hatékonysága**				
	Inverziós	**Betoldási**	**Csere**	**Kétopciós**	**Részútvonalterv**
egl-e1-A	15%	25%	17%	16%	27%
egl-s1-A	16%	23%	16%	16%	29%
egl-g1-A	20%	19%	13%	8%	40%

Megfigyelhető összefüggés a CARP példa mérete és összetettsége, valamint a műveletek hatékonysága között. A probléma méretének növelésével az inverzió és a részútvonalterv műveletek hatékonysága nő a többi művelethez képest, a probléma méretének csökkentésével pedig a betoldási, a csere és a kétopciós műveletek hatékonysága nő a többihez képest.

Az eredmények alapján megállapítható, hogy a részútvonalterv művelet nagyobb valószínűséggel talál jobb megoldást, mint a többi művelet, különösen olyan esetekben, amikor az aktuális megoldáson nagyobb módosításra van szükség (mivel az egy véletlenszerűen generált megoldás és/vagy egy nagyobb CARP példa megoldása).

3.6.3. A felfedezésre összpontosító CARP-ABC algoritmus hatékonyságának vizsgálata CARP példákon

Vizsgálati környezet és paraméterek

A felfedezésre összpontosító CARP-ABC algoritmus (a továbbiakban ABC algoritmus) hatékonyságának vizsgálathoz a D. függelék D.6. szakaszában bemutatott öt különböző CARP példát használtam fel. Az ABC algoritmust, a HMA-t és az ACOPR algoritmust az említett CARP példákra 30-szor és 10 perces futási idő korlátozással futtattam le, egymástól függetlenül. Az egyes algoritmusok végrehajtása során rögzítésre került az új globálisan legjobb megoldás költség értéke és az adott megoldás megtalálásához felhasznált idő (azaz az algoritmus végrehajtása kezdetétől eltelt idő). A rögzített értékek összehasonlításából a következő alszakaszban ismertetett eredmények születtek.

Eredmények

A vizsgálat eredményeit bemutató grafikonok és táblázatok a D. függelék D.7.1. alszakaszában találhatók. A D.2–D.5. ábrákon látható grafikonok az algoritmusok 30 független futtatása során talált globálisan legjobb megoldás költség értékének konvergencia sebességét mutatják a kiválasztott példákra. Az y-tengelyen a megoldás teljes költsége, az x-tengelyen pedig az algoritmus futásának megkezdése óta eltelt idő látható másodpercben. A különböző algoritmusok kimeneteit különböző színek és vonaltípusok jelölik. A színes vonalak az átlagos konvergencia sebességet jelzik, a színes területek pedig az összes rögzített értéket tartalmazzák (azaz a területek a minimális és maximális értékekkel vannak körülhatárolva). Minél közelebb van a vonal a két tengely metszéspontjához, annál jobb az algoritmus konvergencia sebessége.

A **kshs1** példa (D.2. ábra) esetében látható, hogy az ACOPR és az ABC algoritmus konvergencia sebessége körülbelül kétszer olyan gyors, mint a HMA-é. Azonban, annak ellenére, hogy az ACOPR és az ABC algoritmus sebessége közel azonos, az ACOPR algoritmus 30 futtatás során nem találta meg a legjobb megoldást, így az ABC algoritmus a legjobb megoldó a kis méretű CARP példák esetében, mint amilyen a „kshs1" példa is.

Az **egl-e1-A** példa (D.3. ábra) esetében már kezdenek megmutatkozni az algoritmusok konvergencia sebessége közötti különbségek. Látható, hogy az ABC algoritmus minden esetben jobb megoldást ad, és gyorsabban, mint az ACOPR. A HMA-nak nagyon hosszú a ciklusideje, így a konvergencia sebessége is nagyon lassú. Ha az időt nem vesszük figyelembe, a HMA általában jobb megoldásokat tud nyújtani, mint a többi CARP megoldó algoritmus.

A D.5. táblázat az ABC algoritmus és a HMA 30 független futtatása alapján, az algoritmusok által talált globálisan legjobb megoldás teljes költségét vizsgálja különböző időkorlátokon belül. A számított statisztikák a következők: minimum, maximum, átlag és szórás. Az értékek azt mutatják, hogy az ABC algoritmus átlagosan jobb minőségű megoldásokat eredményezett, mint a HMA (azaz kisebb a teljes költség átlag értéke). Ugyanakkor a HMA esetében figyelhető meg a legjobb és a leggyengébb megoldások előfordulása is (azaz a teljes költség minimum értéke kisebb és a maximum értéke nagyobb), ami a teljesítményének nagyobb varianciájára utal. Ennek megfelelően az ABC algoritmus teljesítménye stabilabbnak tekinthető. Érdemes megjegyezni, hogy az ABC és a HMA által produkált megoldások átlagos költsége közötti különbség az idő múlásával fokozatosan csökkent: 1 percen belül körülbelül 200, 5 percen belül megközelítőleg 9, míg 10 percen belül hozzávetőleg 6 volt ez az érték.

Az **egl-s1-A** példa (D.4. ábra és D.6. táblázat) esetében a HMA és az ABC algoritmus konvergencia sebessége közötti különbségek komplexebbek. Látható, hogy 200 másodperc előtt az ABC algoritmus teljesít jobban, 200 és 400 másodperc között az eredmények azonosak, majd 400 másodperc után a HMA teljesít jobban.

Az eredmények hasonlóak voltak az **egl-g1-A** és az **egl-g2-A** példák esetében (D.5. ábra, valamint a D.7. táblázat és a D.8. táblázat). A legtöbb futtatásnál a beállított időkorlát nem volt elegendő ahhoz, hogy a HMA javítsa a kezdeti megoldását, ezért csak a kezdeti megoldás került rögzítésre. Ez az oka annak, hogy a HMA grafikonja egyenes vonalnak tűnik a D.5. ábrán. Következésképpen, a vizsgált időtartamon belül, az ABC algoritmus körülbelül 100 másodperc elteltével jobban teljesített, mint a HMA.

Az eredmények alapján megállapítható, hogy az ABC algoritmus rövid idő alatt

elég jó megoldást tud nyújtani. Mivel kis ciklusidővel rendelkezik, a globálisan legjobb megoldást gyakran tudja frissíteni. Az ABC algoritmus minden szempontból jobb, mint az ACOPR, mivel az ABC algoritmus gyorsabban konvergál, és jobb minőségű megoldásokat talál, mint az ACOPR. Ezenkívül, ha az algoritmusok futási ideje rövid időtartamra van beállítva, a HMA-val versenyképes.

3.6.4. A feltárásra összpontosító CARP-ABC algoritmus és az RR1 algoritmus hatékonyságának vizsgálata az esemény generátort használó DCARP keretrendszer használatával

Vizsgálati környezet és paraméterek

Az esemény generátort használó DCARP keretrendszer használatával elvégzett hatékonyság vizsgálathoz a D. függelék D.6. szakaszában bemutatott közepes méretű CARP példa, az **egl-e1-A** példa lett felhasználva. Mivel a kezdeti DCARP példa alapvetően egy statikus CARP példának tekinthető és a HMA a jelenleg ismert legpontosabb metaheurisztika a CARP megoldásához, ezért az **inicializáló mod**ban a HMA-t használtam a kezdeti megoldás előállításához. Az eseményeket a DCARP keretrendszert ismertető szakaszban bemutatott **esemény létrehozó modul** algoritmusaival (D. függelék D.5.2. alszakasza) generáltam. Minden kritikus eseménytípushoz (azaz a „feladat megjelenés", a „kereslet növekedés" és a „jármű meghibásodás" eseményekhez) 15 esemény lett generálva (egymástól függetlenül), ami a kezdeti példán lett végrehajtva, új DCARP példákat létrehozva.

Az **újratervező modul** által meghívott DCARP megoldóként az RR1 algoritmust, a feltárásra összpontosító ABC algoritmust (a továbbiakban ABC algoritmus) és a HMA-t használtam. Az említett DCARP megoldókat minden esemény bekövetkezésekor 1 perces futási idő korlátozással futtattam le, egymástól függetlenül. A CARP példákon végzett hatékonyság vizsgálathoz hasonlóan, az egyes algoritmusok végrehajtása során rögzítésre került az új globálisan legjobb megoldás költség értéke és az adott megoldás megtalálásához felhasznált idő (azaz az algoritmus végrehajtása kezdetétől eltelt idő). A rögzített értékek összehasonlításából a következő alszakaszban ismertetett eredmények születtek.

Eredmények

A „feladat megjelenés", a „kereslet növekedés" és a „jármű meghibásodás" események bekövetkezésének eredményeit bemutató táblázatok a D. függelék D.7.2. alszakaszában találhatók. Mindhárom táblázatban az első néhány oszlop tartalmazza azokat az attribútum értékeket, amelyekkel rekonstruálhatók a probléma eseményadatainak elemei:

- az utazási és szolgáltatási napló (ts);
- a „feladat megjelenés" esemény (a, dem_a és sc_a);
- a „igény növekedés" esemény (a, dem_a és sc_a);
- a „jármű meghibásodás" esemény (csak h, mivel ekkor h_e értéke mindig 1).

Az RR1 algoritmus, az ABC algoritmus és a HMA által számított legjobb teljes költséget az utolsó három oszlop tartalmazza. Az egyes futások legjobb teljes költségét zöld színnel emeltem ki, a legrosszabbat pedig piros színnel. Ha több algoritmusnak

is ugyanaz a kimenete, akkor mindegyiket kiemeltem.

A **„feladat megjelenés" esemény** esetében (D.9. táblázat) az RR1 és az ABC algoritmusok közel ugyanúgy teljesítettek, a HMA teljesített a legrosszabbul. A **„kereslet növekedés" esemény** esetében (D.10. táblázat) ABC algoritmus teljesített a legjobban és az RR1 algoritmus a legrosszabbul. A HMA teljesítménye körülbelül a kettő között volt. A **„jármű meghibásodás" esemény** esetében az ABC algoritmus teljesített a legjobban. Az RR1 algoritmus, valamint a HMA körülbelül ugyanúgy teljesítettek, de a HMA valamivel jobban.

Az összes eseményre végzett vizsgálat eredményeit a 3.4. táblázat foglalja össze. A táblázatban az algoritmusok kimenetei vannak pontozva eseménytípusonként. Az értékek úgy lettek kiszámolva, hogy eseménytípusonként és algoritmusonként összegezve lett a futások eredményére adható pontszám. Futásonként a futás legjobb kimenete 1 pontot ér, a legrosszabb pedig -1-et. Látható, hogy összességében az ABC algoritmus teljesített a legjobban és az RR1 algoritmus a legrosszabbul.

3.4. táblázat. Az RR1, az ABC és a HMA kimenetek pontozása az „egl-e1-A" példa esetében, minden egyes eseménytípusra

Esemény típusa	**Algoritmus kimenetei**		
	RR1	**ABC**	**HMA**
feladat megjelenés	7	7	-2
kereslet növekedés	-14	11	4
jármű meghibásodás	1	7	3
Összesen	-6	25	5

A táblázatokban nem látható, de az RR1 algoritmus futási ideje a legrövidebb (a tesztesetekben mindig kevesebb, mint egy másodperc volt). A többi algoritmus (ABC és HMA) futási ideje megközelítőleg azonos, akár DCARP, akár CARP példa szolgál bemenetként, mivel elsősorban a probléma komplexitása határozza meg a konvergencia sebességét.

Az eredmények alapján a CARP példákon végzett hatékonyság vizsgálathoz hasonló következtetések vonhatók le. Az ABC algoritmus egy bizonyos ideig felülmúlja a HMA-t, de aztán a HMA lassan átveszi a vezetést. Ha az idő a prioritás, akkor a „feladat megjelenés" esemény bekövetkezése esetén az RR1 algoritmust kell használni. Ha az idő és a megoldás minősége egyaránt fontos, akkor az ABC algoritmust kell alkalmazni bármely esemény bekövetkezése esetén. Ha viszont a megoldás minősége az elsődleges és nem vagyunk időkorláthoz kötve, akkor a HMA-t érdemes használni.

3.6.5. A feltárásra összpontosító CARP-ABC algoritmus és az RR1 algoritmus hatékonyságának vizsgálata a forgalom szimulációt használó DCARP keretrendszer használatával

Vizsgálati környezet és paraméterek

A forgalom szimulációt használó DCARP keretrendszer használatával elvégzett hatékonyság vizsgálatokhoz a D. függelék D.6. szakaszában bemutatott 12 DCARP forgatókönyvet használtam fel. Pontosabban minden forgatókönyvnek csak a kezdeti példáját és annak megoldását alkalmaztam, mivel a rákövetkező példák az alkalmazott (D)CARP megoldó algoritmus által biztosított szolgáltatási terv függvényében változnak.

Mind a 12 DCARP forgatókönyvhöz háromféle vizsgálatot végeztem el:

1. csak a járműforgalmi szimulátor által generált forgalomváltozási események (azaz „forgalom növekedés" és „forgalom csökkenés") következnek be
2. a forgalmi változásokon kívül 10 „feladat megjelenés" esemény következik be
3. a forgalmi változásokon kívül 20 „feladat megjelenés" esemény következik be

Mindhárom vizsgálat során a teljes költség (ami ebben az esetben a teljes szolgáltatási időt jelenti) és a szimulációs idő hossza került rögzítésre, másodpercben mérve. Mivel itt fontos, hogy újratervezés esetén minél előbb rendelkezésre álljon egy végrehajtható megoldás, mindegyik algoritmushoz 1 perces futási időkorlát lett beállítva. A determinisztikus algoritmusok esetében (RR1 és HyLS) a szimuláció egyszer került lefuttatásra, mivel azok mindig ugyanazt a kimenetet biztosítják ugyanarra a bemenetre. A nem determinisztikus algoritmusok esetében (feltárásra összpontosító ABC algoritmus és HMA) a szimuláció 10 alkalommal került lefuttatásra.

Az első vizsgálatnál a HyLS algoritmust és az ABC algoritmust alkalmaztam DCARP optimalizáló algoritmusként. Mivel a forgalomváltozási események nem módosítják a szolgáltatási terv megvalósíthatóságát, ezért úgy is lefuttattam a szimulációkat, hogy semmilyen DCARP optimalizáló algoritmus sem került felhasználásra. A másik kettő vizsgálatnál „feladat megjelenés" események is vannak, amik bekövetkezése esetén mindig szükséges a szolgáltatási terv frissítése, ezért ennél a két vizsgálatnál muszáj volt DCARP optimalizáló algoritmust használnom. Az RR1 és az ABC algoritmusokat mindkét vizsgálatnál alkalmaztam. Ezeken kívül a másodiknál a HyLS, a harmadiknál pedig a HMA került alkalmazásra.

Eredmények

A vizsgálatok részletes eredményeit bemutató grafikonok és táblázatok a D. függelék D.7.3. szakaszában találhatók. Mindhárom vizsgálat teljes költségre és szimuláció időtartamra vonatkozó eredményeit rendre a 3.5. táblázat és a 3.6. táblázat foglalja össze. A táblázatokban az algoritmusok kimenetei vannak pontozva vizsgálatonként, teljes költség és szimuláció időtartam szerint. Az első oszlop a vizsgálat sorszámát tartalmazza. Az értékek úgy lettek kiszámolva, hogy vizsgálatonként és algoritmusonként (vagy algoritmus kombinációnként) összegezve lettek a DCARP forgatókönyvenként elért kimenetekre adható pontszámok. DCARP forgatókönyvenként a legjobb kimenet 1 pontot ér, a legrosszabb pedig -1-et. A részletes és az

összegzett eredményeket bemutató táblázatokban soronként a legjobb értéket zöld, a legrosszabbat pedig piros színnel emeltem ki. Ha több algoritmusnak is ugyanaz a kimenete, akkor mindegyiket kiemeltem. A szürke szín azt jelzi, hogy az adott algoritmus (vagy algoritmus kombináció) nem lett alkalmazva a vizsgálat során.

3.5. táblázat. A DCARP megoldó algoritmusok kimeneteinek teljes költség szerinti pontozása mindhárom vizsgálat esetében

#	Algoritmus kimenetei						
	nincs	**RR1**	**HyLS**	**ABC**	**RR1 & ABC**	**HMA**	**RR1 & HMA**
1	-6		2	5			
2		4	-3	-1			
				0	6	-5	-1
3		0		3		-3	

3.6. táblázat. A DCARP megoldó algoritmusok kimeneteinek szimuláció időtartama szerinti pontozása mindhárom vizsgálat esetében

#	Algoritmus kimenetei						
	nincs	**RR1**	**HyLS**	**ABC**	**RR1 & ABC**	**HMA**	**RR1 & HMA**
1	-3		2	2			
2		3	-1	0			
				-2	5	-6	3
3		1		3		-4	

Az **első vizsgálat** teljes költségre vonatkozó eredményeit a D.12. táblázat, a szimuláció időtartamára vonatkozót pedig a D.13. táblázat foglalja össze. Az eredmények azt mutatják, hogy a legtöbb forgatókönyv esetében akkor a legmagasabb a teljes költség, ha semmilyen DCARP optimalizáló sincsen használva. Ennélfogva, az áthaladási költségek túlságos megnövekedése esetén javasolt az aktuális szolgáltatási terv újratervezése még akkor is, ha az még megvalósítható. A HyLS és az ABC algoritmusok átlagosan körülbelül ugyanúgy teljesítettek, ha csak a szimulációk időtartamát nézzük. Ugyanakkor az ABC algoritmus szinte minden (egy kivételével) forgatókönyv esetén a legalacsonyabb teljes költségű szolgáltatási terveket találta meg, így valamivel jobbnak tekinthető.

A **második vizsgálat** során az RR1, az ABC és a HyLS algoritmusok alkalmazása esetén mért teljes költségre vonatkozó eredmények a D.14. táblázat, a szimuláció időtartamára vonatkozó pedig a D.15. táblázat foglalja össze. Az eredmények azt mutatják, hogy a legtöbb forgatókönyv esetében az RR1 algoritmus teljesített a legjobban, a HyLS algoritmus pedig a legrosszabbul. Az ABC algoritmus átlagosan kevesebb forgatókönyvnél eredményezte a legalacsonyabb összköltséget, mint a HyLS algoritmus, de ez a legmagasabb összköltségre is elmondható. Ebből kifolyólag az ABC algoritmus valamivel hatékonyabbnak tekinthető, mint a HyLS algoritmus.

A második vizsgálat során az ABC, az RR1 & ABC, a HMA és az RR1 & HMA algoritmus kombinációk alkalmazása esetén mért teljes költségre vonatkozó eredmények a D.6. ábra és a D.16. táblázat, a szimuláció időtartamára vonatkozó pedig a D.7. ábra és D.17. táblázat foglalja össze. Láthatjuk, hogy a 6. és a 12. forgatókönyv esetében a legkevésbé szórtak a teljes költség értékek, a 3. és a 9. forgatókönyv esetében pedig a legjobban. Összességében elmondható, hogy 10 futtatás átlagértéke jó becslést adhat a teljes költség átlagos értékre. Láthatjuk, hogy szinte minden forgatókönyv esetében a szimuláció időtartam értékek nagyobb szórást mutatnak, mint a teljes költségek esetében, valamint több olyan érték is van, amely kiugró értéknek tekinthető. Ezen okok miatt a 10 futtatás átlagértéke valószínűleg nem ad olyan jó becslést, mint a teljes költség esetében. A táblázatokból az olvasható ki, hogy a legtöbb forgatókönyv esetében az RR1 & ABC algoritmus kombinációja teljesített a legjobban, a HMA pedig a legrosszabbul.

A **harmadik vizsgálat** teljes költségre vonatkozó eredményeit a D.18. táblázat, a szimuláció időtartamára vonatkozót pedig D.19. táblázat foglalja össze. Az eredmények azt mutatják, hogy átlagosan az ABC algoritmus teljesített a legjobban, a HMA pedig a legrosszabbul. A HMA esetében látható, hogy az 1. és a 6. forgatókönyvnél nagyon magas értékek is voltak, így a szórás is nagy. Ennek az lehet az oka, hogy egy vagy több jármű forgalmi dugóba került. Egy másik lehetséges ok, hogy az egyik új feladatot csak egy új járművel lehetett kiszolgálni.

Összességében megállapítható, hogy az RR1 és az ABC algoritmus kombinációja adhatja a legjobb szolgáltatási terveket. Ezért a megvalósíthatatlanná vált szolgáltatási tervet először az RR1 algoritmusnak ajánlott átadni, hogy azt megvalósíthatóvá javítsa, majd további finomítás céljából a terv átadható az ABC algoritmusnak.

3.7. Összefoglalás

Ebben a fejezetben a statikus és dinamikus kapacitáskorlátos ív útvonaltervezési problémához (CARP és DCARP) általam kifejlesztett megoldásokat mutattam be és teszteltem. Ezek között szerepel egy (D)CARP-hoz használható új mozgási művelet (részútvonalterv művelet), két új (D)CARP megoldó ABC algoritmus (egy felfedezésre összpontosító és egy feltárásra összpontosító változat), valamint egy adat-vezérelt DCARP keretrendszer, amiből a hatékonyság vizsgálatok elvégzéséhez két változatot is készítettem (egy esemény generátort használó és egy forgalom szimulációt használó változat).

A javasolt részútvonalterv művelet

A javasolt részútvonalterv művelet egy összetett, közepes lépésméretű mozgási művelet, amely két különböző mohó keresési módszert – a mohó újrakapcsolási és a részútvonalterv forgatási módszert – valamint egy torzítást biztosító módszert foglal magába. Egyszerre csak egy útvonalterv feladatainak sorrendjét és kiszolgálási irányát módosíthatja, így a szolgáltatási terv megvalósíthatóságát biztos nem módosítja. A művelet hatékonyságát különböző bonyolultsági szintű CARP példákon és az irodalomban fellelhető (CARP-hoz használt) mozgási műveletek hatékonyságához viszonyítva mértem. Az eredmények azt mutatták, hogy a részútvonalterv művelet nagyobb valószínűséggel talál jobb megoldást, mint a többi művelet.

A javasolt CARP-ABC algoritmusok

A CARP-ABC algoritmusok rajintelligencia alapú optimalizáló algoritmusok, amiket a mézelő méhraj intelligens táplálékkereső viselkedése inspirált. Az algoritmusokból összesen két változatot készítettem: egy felfedezésre és egy feltárásra összpontosító változatot. A felfedezésre összpontosító változat célja, hogy minél változatosabb megoldásokat biztosítson, ezért azt statikus CARP megoldóként ajánlott használni. Evvel szemben a feltárásra összpontosító változat célja, hogy minél hamarabb találjon minél jobb megoldást, ezért azt DCARP megoldóként javasolt alkalmazni.

A CARP-ABC algoritmus felfedezésre összpontosító változatának a hatékonyságát különböző bonyolultsági szintű CARP példákon és az irodalomban fellelhető CARP megoldó algoritmusok (HMA és ACOPR) hatékonyságához viszonyítva mértem. Az eredmények azt mutatták, hogy az algoritmus versenyképesnek tekinthető a jelenleg legpontosabb CARP megoldóval, a HMA-val szemben, amikor az algoritmusok maximum megengedett futási ideje körülbelül egy percre van korlátozva.

A CARP-ABC algoritmus feltárásra összpontosító változata az adat-vezérelt DCARP keretrendszer használatával lett tesztelve.

A javasolt adat-vezérelt DCARP keretrendszer és az RR1 újratervező algoritmus

Az adat-vezérelt DCARP keretrendszer minden lehetséges eseményt képes kezelni. A keretrendszer feladata, hogy a megfigyelt eseményeket feldolgozva aktualizálja a problémát (új DCARP példát generálva) és a szolgáltatási tervet. Egy kritikus esemény bekövetkezése esetén egy (D)CARP megoldó algoritmust hív segítségül a szolgáltatási terv újratervezéséhez. A keretrendszer részeként egy minimális újratervező (RR1) algoritmust is létrehoztam, amely célja, hogy a megvalósíthatatlanná vált szolgáltatási tervet megvalósíthatóvá tegye, minimális számú útvonalterv módosítás mellett.

A DCARP megoldó algoritmusok (a feltárásra összpontosító CARP-ABC algoritmus és az RR1 algoritmus) valósághű környezetben való teszteléséhez a DCARP keretrendszerből két változatot is készítettem, amelyek főként a szimulációhoz használt adatok előállításában különböznek. Az esemény generátort használó változat egy esemény létrehozó modult használ minden esemény generálásához. A forgalom szimulációt használó változat egy forgalom szimulációs szoftvert használ az utazási és szolgáltatási események generálásához, valamint a forgalom változást (forgalom növekedést vagy csökkenést) leíró események létrehozásához. A többi eseményhez ez a változat is az esemény létrehozó modult használja.

A CARP-ABC algoritmus feltárásra összpontosító változatának a hatékonyságát, valamint az RR1 algoritmus hatékonyságát az adat-vezérelt DCARP keretrendszer mindkét változatával teszteltem. Az esemény generátort használó változat esetében egy valós CARP példát használtam fel, a forgalom szimulációt használó változat esetében pedig egy valós úthálózat és ahhoz tartozó forgalmi adatokat (konkrétan az azokból generált forgatókönyveket). Mindkét változat esetén a hatékonyság elemzéshez az irodalomban fellelhető (D)CARP megoldó algoritmusok (HMA és HyLS) is felhasználásra kerültek. Az eredmények azt mutatták, hogy átlagban az RR1 és a feltárásra összpontosító CARP-ABC algoritmus együttes használata adja a legjobb szolgáltatási tervet.

4. fejezet

Összefoglalás

A doktori értekezésemben üzleti folyamatok online nyomon követését és javítását lehetővé tevő, folyamatbányászati és mesterséges intelligencia algoritmusokat alkalmazó megoldásokat fejlesztettem ki.

A dolgozatom első felében elsősorban a rövid átfutási idővel rendelkező folyamatok (pl. gyártási folyamatok) online megfigyelését és javítását támogató megoldások létrehozására koncentráltam. Ennek keretében a folyamatbányászat egyik fő típusával, a megfelelőség-ellenőrzéssel, valamint folyamatadat vizualizációval foglalkoztam. Kifejlesztettem az első (előtag-igazítás) alapú MOCC módszert, ami egy több perspektívás folyamatmodell és egy eseményfolyam megfigyelt eseményei közötti optimális több perspektívás előtag-igazításokat ad vissza. Ezenfelül létrehoztam két folyamatadat vizualizációs módszert is: egyet az eredeti eseményadatokhoz, és egyet az MOCC módszer kimeneteihez, azaz igazítási adatokhoz. A módszerek egy Gantt-diagram által inspirált vegyes idővonal-alapú vizualizációt használnak, amik – a jelenleg elterjedt vizualizációs megoldásokkal ellentétben – megfelelően kezelik az átfedő objektumokat. A valós folyamatokon végzett vizsgálatok eredményei azt mutatták, hogy a javasolt megoldások együttes használata lehetővé teszi a folyamatok hatékony online nyomonkövetését. Használatukkal könnyen kimutathatók az amúgy nehezen fellelhető rendellenségek és azok lehetséges kiváltó okai is, ezáltal segítve a folyamat működésének javítását. A jövőben ez még kiegészíthető egy ajánlórendszerrel, ami a detektált rendellenességek alapján konkrét javító tevékenységeket tud javasolni a folyamat javítása érdekében. Továbbá, a megoldások valós környezetben történő tesztelésére is szükség van még.

A dolgozatom második felében a szállítási folyamatok online megfigyelését és javítását támogató megoldások létrehozására koncentráltam. Ennek keretében a kapacitáskorlátos ív útvonaltervezési problémával (CARP) és annak dinamikus változatával (DCARP) foglalkoztam. A (D)CARP-hoz létrehoztam egy közepes lépésméretű mozgási műveletet, a részútvonalterv műveletet, ami nagyobb valószínűséggel talál jobb megoldást, mint a jelenleg létező kis lépésméretű műveletek. Ezenfelül kifejlesztettem az első ABC algoritmust a CARP megoldására (CARP-ABC algoritmus), amely – a jelenleg létező mozgási műveleteken kívül – a részútvonalterv műveletet is alkalmazza a keresési folyamatai során. A CARP-ABC algoritmusból összesen két változatot készítettem: (CARP-hoz) egy felfedezésre és (DCARP-hoz) egy feltárásra összpontosító változatot. Emellett létrehoztam egy adat-vezérelt DCARP keretrendszert, ami minden lehetséges eseményt képes kezelni. A keretrendszer a megfigyelt eseményeket feldolgozva aktualizálja a problémát és a szolgáltatási tervet, és újra-

tervezés szükségessége esetén egy (D)CARP megoldó algoritmust hív segítségül. A keretrendszer részeként egy minimális újratervező (RR1) algoritmust is létrehoztam, amely célja, hogy a megvalósíthatatlanná vált szolgáltatási tervet megvalósíthatóvá tegye, minimális számú útvonalterv módosítás mellett. Az elvégzett vizsgálatok eredményei azt mutatták, hogy CARP példák esetében a CARP-ABC algoritmus felfedezésre összpontosító változata rövid idő alatt – a jelenleg létező CARP megoldókhoz viszonyítva – elég jó megoldást tud nyújtani, DCARP példák esetében pedig átlagban az RR1 és a CARP-ABC algoritmus feltárásra összpontosító változatának együttes használata adja a legjobb szolgáltatási tervet. A jövőben a (D)CARP megoldók bonyolultabb forgatókönyveken is tesztelésre kerülnek majd.

Irodalomjegyzék

[1] W. Van Der Aalst, *Process mining: discovery, conformance and enhancement of business processes*. Springer Berlin, Heidelberg, 2011, 1. köt. DOI: https://doi.org/10.1007/978-3-642-19345-3.

[2] W. Van Der Aalst, *Process mining: data science in action*. Springer Berlin, Heidelberg, 2016, 2. köt. DOI: https://doi.org/10.1007/978-3-662-49851-4.

[3] W. M. van der Aalst és J. Carmona, *Process mining handbook*. Springer, Cham, 2022, 1. köt. DOI: https://doi.org/10.1007/978-3-031-08848-3.

[4] J. Carmona, B. van Dongen, A. Solti és M. Weidlich, *Conformance checking*. Springer, Cham, 2018, 1. köt. DOI: https://doi.org/10.1007/978-3-319-99414-7.

[5] S. J. van Zelst, A. Bolt és B. F. van Dongen, „Tuning Alignment Computation: An Experimental Evaluation", *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2017 (ATAED 2017), Zaragoza, Spain, June 26–27, 2017*, 1847. köt., CEUR-WS.org, Aachen, Germany, 2017, 6–20. old. cím: https://ceur-ws.org/Vol-1847/paper01.pdf.

[6] S. J. van Zelst, A. Bolt, M. Hassani, B. F. van Dongen és W. M. van der Aalst, „Online conformance checking: relating event streams to process models using prefix-alignments", *International Journal of Data Science and Analytics*, 8. évf., 269–284. old., 2019. DOI: https://doi.org/10.1007/s41060-017-0078-6.

[7] D. Schuster és S. J. van Zelst, „Online process monitoring using incremental state-space expansion: an exact algorithm", *Business Process Management: 18th International Conference, BPM 2020, Seville, Spain, September 13–18, 2020*, Springer, Cham, 2020, 147–164. old. DOI: https://doi.org/10.1007/978-3-030-58666-9_9.

[8] A. Burattin és J. Carmona, „A framework for online conformance checking", *Business Process Management Workshops: BPM 2017 International Workshops, Barcelona, Spain, September 10–11, 2017*, Springer, Cham, 2018, 165–177. old. DOI: https://doi.org/10.1007/978-3-319-74030-0_12.

[9] A. Burattin, S. J. van Zelst, A. Armas-Cervantes, B. F. van Dongen és J. Carmona, „Online conformance checking using behavioural patterns", *Business Process Management: 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9–14, 2018*, Springer, Cham, 2018, 250–267. old. DOI: https://doi.org/10.1007/978-3-319-98648-7_15.

[10] W. L. J. Lee, A. Burattin, J. Munoz-Gama és M. Sepúlveda, „Orientation and conformance: A HMM-based approach to online conformance checking”, *Information Systems*, 102. évf., 2021. DOI: https://doi.org/10.1016/j.is.2020.101674.

[11] M. De Leoni, W. M. Van Der Aalst és B. F. Van Dongen, „Data-and resource-aware conformance checking of business processes”, *Business Information Systems: 15th International Conference, BIS 2012, Vilnius, Lithuania, May 21–23, 2012*, Springer, Berlin, Heidelberg, 2012, 48–59. old. DOI: https://doi.org/10.1007/978-3-642-30359-3_5.

[12] M. De Leoni és W. M. Van Der Aalst, „Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming”, *Business Process Management: 11th International Conference, BPM 2013, Beijing, China, August 26–30, 2013*, Springer, Berlin, Heidelberg, 2013, 113–129. old. DOI: https://doi.org/10.1007/978-3-642-40176-3_10.

[13] F. Mannhardt, M. De Leoni, H. A. Reijers és W. M. Van Der Aalst, „Balanced multi-perspective checking of process conformance”, *Computing*, 98. évf., 407–437. old., 2016. DOI: https://doi.org/10.1007/s00607-015-0441-1.

[14] **Zs. Nagy**, Á. Werner-Stark és T. Dulai, „An industrial application using process mining to reduce the number of faulty products”, *New Trends in Databases and Information Systems: ADBIS 2018 Short Papers and Workshops, AI* QA, BIGPMED, CSACDB, M2U, BigDataMAPS, ISTREND, DC, Budapest, Hungary, September 2–5, 2018, Proceedings*, Springer, Cham, 2018, 352–363. old. DOI: https://doi.org/10.1007/978-3-030-00063-9_33.

[15] **Zs. Nagy**, Á. Werner-Stark és T. Dulai, „Analysis of industrial logs to reduce the number of faulty products of manufacturing”, *OGIK 2018: 15. Országos Gazdaságinformatikai Konferencia - Az előadások összefoglalói, Sopron, Hungary, November 9–10, 2018*, L. Bacsárdi, G. Bencsik és Z. Pödör, szerk., Sopron, Hungary: Alexander Alapítvány a Jövő Értelmiségéért, 2018, 26–27. old.

[16] **Zs. Nagy**, Á. Werner-Stark és T. Dulai, „Analysis of industrial logs to reduce the number of faulty products of manufacturing”, *OGIK’2018 Országos Gazdaságinformatikai Konferencia - Válogatott közlemények, Sopron, Hungary, November 9–10, 2018*, L. Bacsárdi, G. Bencsik és Z. Pödör, szerk., Sopron, Hungary: Alexander Alapítvány a Jövő Értelmiségéért, 2019, 53–57. old.

[17] **Zs. Nagy**, A. Werner-Stark és T. Dulai, „Using process mining in real-time to reduce the number of faulty products”, *Advances in Databases and Information Systems: 23rd European Conference, ADBIS 2019, Bled, Slovenia, September 8–11, 2019, Proceedings*, T. Welzer, J. Eder, V. Podgorelec és A. K. Latifić, szerk., Lecture Notes in Computer Science sor., 11695. köt., Springer, Cham, 2019, 89–104. old. DOI: https://doi.org/10.1007/978-3-030-28730-6_6.

[18] J. Carmona, B. van Dongen és M. Weidlich, „Conformance checking: foundations, milestones and challenges”, *Process Mining Handbook*, Springer, Cham, 2022, 155–190. old. DOI: https://doi.org/10.1007/978-3-031-08848-3_5.

[19] N. Assy, C. Souchet, T. Bouffard és O. Anesini, „Visualization Libraries for Process Analytics”, *Proceedings of the ICPM Doctoral Consortium and Demo Track 2022 (ICPM-D 2022) co-located with 4th International Conference on Process Mining (ICPM 2022), Bolzano, Italy, October 23–28, 2022*, M. Hassani, A. Koschmider, M. Comuzzi, F. M. Maggi és L. Pufahl, szerk., 3299. köt., 2022, 80–84. old. cím: https://ceur-ws.org/Vol-3299/Paper17.pdf.

[20] **Zs. Nagy** és Á. Werner-Stark, „A Multi-perspective Online Conformance Checking Technique”, *2020 6th International Conference on Information Management (ICIM), London, UK, March 27–29, 2020, Proceedings*, IEEE, New York, NY, USA, 2020, 172–176. old. DOI: https://doi.org/10.1109/ICIM49319.2020.244693.

[21] **Zs. Nagy** és Á. Werner-Stark, „An alignment-based multi-perspective online conformance checking technique”, *Abstract book for the 16th MIKLÓS IVÁNYI INTERNATIONAL PHD & DLA SYMPOSIUM, Pécs, Hungary, October 26–27, 2020*, P. Iványi, szerk., Pécs, Hungary: Pollack Press, 2020, 126. old.

[22] **Zs. Nagy** és A. Werner-Stark, „An Alignment-based Multi-Perspective Online Conformance Checking Technique”, *Acta Polytechnica Hungarica*, 19. évf., 4. sz., 105–127. old., 2022. cím: http://acta.uni-obuda.hu/Nagy_Werner-Stark_122.pdf.

[23] **Zs. Nagy** és A. Werner-Stark, „Visualization Methods to Support Real-time Process Monitoring”, *Proceedings of the 3rd International Workshop on Information Technologies: Theoretical and Applied Problems 2023 (ITTAP 2023), Ternopil, Ukraine, Opole, Poland, November 22–24, 2023*, 3628. köt., CEUR-WS.org, Aachen, Germany, 2023, 1–14. old. cím: https://ceur-ws.org/Vol-3628/paper1.pdf.

[24] B. L. Golden és R. T. Wong, „Capacitated arc routing problems”, *Networks*, 11. évf., 3. sz., 305–315. old., 1981. DOI: https://doi.org/10.1002/net.3230110308.

[25] V. Maniezzo, „Algorithms for large directed CARP instances: urban solid waste collection operational support”, Department of Computer Science, University of Bolonha, Mura Anteo Zamboni 7, 40127 Bologna, Italy, techn. jel. UBLCS-2004-16, 2004. okt.

[26] E. Babaee Tirkolaee, I. Mahdavi, M. M. Seyyed Esfahani és G.-W. Weber, „A hybrid augmented ant colony optimization for the multi-trip capacitated arc routing problem under fuzzy demands for urban solid waste management”, *Waste management & research*, 38. évf., 2. sz., 156–172. old., 2020. DOI: https://doi.org/10.1177/0734242X19865782.

[27] R. W. Eglese, „Routeing winter gritting vehicles”, *Discrete applied mathematics*, 48. évf., 3. sz., 231–244. old., 1994. DOI: https://doi.org/10.1016/0166-218X(92)00003-5.

[28] J. Fink, M. Loebl és P. Pelikánová, „Arc-routing for winter road maintenance”, *Discrete optimization*, 41. évf., 100644. old., 2021. DOI: https://doi.org/10.1016/j.disopt.2021.100644.

[29] X. Wang és E. Wasil, „On the road to better routes: Five decades of published research on the vehicle routing problem”, *Networks*, 77. évf., 1. sz., 66–87. old., 2021. DOI: https://doi.org/10.1002/net.21942.

[30] A. Corberán és C. Prins, „Recent results on arc routing problems: An annotated bibliography”, *Networks*, 56. évf., 1. sz., 50–69. old., 2010. DOI: https://doi.org/10.1002/net.20347.

[31] M. C. Mourão és L. S. Pinto, „An updated annotated bibliography on arc routing problems”, *Networks*, 70. évf., 3. sz., 144–194. old., 2017. DOI: https://doi.org/10.1002/net.21762.

[32] Á. Corberán, R. Eglese, G. Hasle, I. Plana és J. M. Sanchis, „Arc routing problems: A review of the past, present, and future”, *Networks*, 77. évf., 1. sz., 88–115. old., 2021. DOI: https://doi.org/10.1002/net.21965.

[33] K. Tang, Y. Mei és X. Yao, „Memetic algorithm with extended neighborhood search for capacitated arc routing problems”, *IEEE Transactions on Evolutionary Computation*, 13. évf., 5. sz., 1151–1166. old., 2009. DOI: https://doi.org/10.1109/TEVC.2009.2023449.

[34] H. Fu, Y. Mei, K. Tang és Y. Zhu, „Memetic algorithm with heuristic candidate list strategy for capacitated arc routing problem”, *IEEE Congress on Evolutionary Computation, Barcelona, Spain, July 18–23, 2010*, IEEE, New York, NY, USA, 2010, 1–8. old. DOI: https://doi.org/10.1109/CEC.2010.5586042.

[35] X. Chen, „Maens+: A divide-and-conquer based memetic algorithm for capacitated arc routing problem”, *2011 Fourth International Symposium on Computational Intelligence and Design, Hangzhou, China, October 28–30, 2011*, IEEE, New York, NY, USA, 1. köt., 2011, 83–88. old. DOI: https://doi.org/10.1109/ISCID.2011.30.

[36] Y. Chen, J.-K. Hao és F. Glover, „A hybrid metaheuristic approach for the capacitated arc routing problem”, *European Journal of Operational Research*, 253. évf., 1. sz., 25–39. old., 2016. DOI: https://doi.org/10.1016/j.ejor.2016.02.015.

[37] C.-J. TING és H.-S. TSAI, „Ant colony optimization with path relinking for the capacitated arc routing problem”, *Asian Transport Studies*, 5. évf., 2. sz., 362–377. old., 2018. DOI: https://doi.org/10.11175/eastsats.5.362.

[38] D. Karaboga, „An idea based on honey bee swarm for numerical optimization”, Computer Engineering Department, Engineering Faculty, Erciyes University, Kayseri, Turkey, techn. jel. tr06, 2005. okt.

[39] D. Karaboga és B. Gorkemli, „A combinatorial artificial bee colony algorithm for traveling salesman problem”, *2011 International Symposium on Innovations in Intelligent Systems and Applications, Istanbul, Turkey, June 15–18, 2011*, IEEE, New York, NY, USA, 2011, 50–53. old. DOI: https://doi.org/10.1109/INISTA.2011.5946125.

[40] B. Görkemli és D. Karaboga, „Quick combinatorial artificial bee colony-qCABC-optimization algorithm for TSP”, *2nd International Symposium on Computing in Informatics and Mathematics (ISCIM 2013), Tirana, Albania, September 26, 2013*, Epoka University, Tirana, Albania, 2013, 50–53. old. cím: http://dspace.epoka.edu.al/handle/1/849.

[41] T. Cura, „An artificial bee colony approach for the undirected capacitated arc routing problem with profits”, *International Journal of Operational Research*, 17. évf., 4. sz., 483–508. old., 2013. DOI: https://doi.org/10.1504/IJOR.2013.054973.

[42] D. Karaboga és B. Gorkemli, „Solving traveling salesman problem by using combinatorial artificial bee colony algorithms”, *International Journal on Artificial Intelligence Tools*, 28. évf., 1. sz., 2019. DOI: https://doi.org/10.1142/S0218213019500040.

[43] K. Kantawong és S. Pravesjit, „An enhanced ABC algorithm to solve the vehicle routing problem with time windows”, *ECTI Transactions on Computer and Information Technology (ECTI-CIT)*, 14. évf., 1. sz., 46–52. old., 2020. DOI: https://doi.org/10.37936/ecti-cit.2020141.200016.

[44] S. Mortada és Y. Yusof, „A Neighbourhood Search for Artificial Bee Colony in Vehicle Routing Problem with Time Windows.”, *International Journal of Intelligent Engineering & Systems*, 14. évf., 3. sz., 255–266. old., 2021. DOI: https://doi.org/10.22266/ijies2021.0630.22.

[45] G. Wu, K. Zhao, J. Cheng és M. Ma, „A Coordinated Vehicle–Drone Arc Routing Approach Based on Improved Adaptive Large Neighborhood Search”, *Sensors*, 22. évf., 10. sz., 2022. DOI: https://doi.org/10.3390/s22103702.

[46] H. Tong, L. L. Minku, S. Menzel, B. Sendhoff és X. Yao, „A Novel Generalized Metaheuristic Framework for Dynamic Capacitated Arc Routing Problems”, *IEEE Transactions on Evolutionary Computation*, 26. évf., 6. sz., 1486–1500. old., 2022. DOI: https://doi.org/10.1109/TEVC.2022.3147509.

[47] T. Dulai, Á. Werner-Stark és K. M. Hangos, „Algorithm for directing cooperative vehicles of a vehicle routing problem for improving fault-tolerance”, *optimization and Engineering*, 19. évf., 239–270. old., 2018. DOI: https://doi.org/10.1007/s11081-017-9353-6.

[48] **Zs. Nagy**, Á. Werner-Stark és T. Dulai, „Solving the Capacitated Arc Routing Problem by a Special Evolutionary Optimization Algorithm”, *OGIK’2021 Országos Gazdaságinformatikai Konferencia, Veszprém, Hungary, November 12–13, 2021*, Veszprém, Hungary: Platina Nyomda és Kiadó Kft., 2021, 41–42. old.

[49] **Zs. Nagy**, A. Werner-Stark és T. Dulai, „An Artificial Bee Colony Algorithm for Static and Dynamic Capacitated Arc Routing Problems”, *Mathematics*, 10. évf., 13. sz., 2022. DOI: https://doi.org/10.3390/math10132205. cím: https://www.mdpi.com/2227-7390/10/13/2205.

[50] **Zs. Nagy**, Á. Werner-Stark és T. Dulai, „Solving Data-driven Dynamic Capacitated Arc Routing Problems”, *Middle-European Conference on Applied Theoretical Computer Science (MATCOS-22), Koper, Slovenia, October 13–14, 2022, Booklet*, 2022, 16. old. cím: https://matcos22.iam.upr.si/en/resources/files/published-material/matcos-22-booklet.pdf.

[51] **Zs. Nagy**, Á. Werner-Stark és T. Dulai, „Comparison of Optimization Algorithms for the Dynamic Capacitated Arc Routing Problem”, *Abstracts of the International Conference on Optimization and Algorithms (OPAL 2023) Semi Online 25, Veszprém, Hungary, June 5–9, 2023*, 2023, 19. old. cím: https://mik.uni-pannon.hu/docs/conferences/opal2023/Abstacts_of_OPAL_15.pdf.

[52] **Zs. Nagy**, Á. Werner-Stark és T. Dulai, „A Data-driven Solution for the Dynamic Capacitated Arc Routing Problem”, *Proceedings of IAC in Budapest 2021, Budapest, Hungary, November 26–27, 2021*, H. Kratochvílová és R. Kratochvíl, szerk., Prague, Czech Republic: Czech Institute of Academic Education z.s., 2021, 64–83. old. cím: https://www.conferences-scientific.cz/file/9788088203247.

[53] A. Rozinat és W. M. van der Aalst, „Conformance testing: measuring the alignment between event logs and process models”, BETA publicatie: working papers sor., 144. köt., Technische Universiteit Eindhoven, 2005. cím: https://pure.tue.nl/ws/portalfiles/portal/2250715/596570.pdf.

[54] M. Pesic, H. Schonenberg és W. van der Aalst, „Declarative workflow”, *Modern business process automation: YAWL and its support environment*, Springer, Berlin, Heidelberg, 2009, 175–201. old. DOI: https://doi.org/10.1007/978-3-642-03121-2_6.

[55] A. Adriansyah, B. F. van Dongen és W. M. van der Aalst, „Conformance checking using cost-based fitness analysis”, *2011 IEEE 15th international enterprise distributed object computing conference, Helsinki, Finland, August 29 – September 2, 2011*, IEEE, New York, NY, USA, 2011, 55–64. old. DOI: https://doi.org/10.1109/EDOC.2011.12.

[56] S. Dunzer, M. Stierle, M. Matzner és S. Baier, „Conformance checking: a state-of-the-art literature review”, *S-BPM ONE '19: Proceedings of the 11th international conference on subject-oriented business process management, Seville, Spain, June 26–28, 2019*, ACM, New York, NY, USA, 2019, 1–10. old. DOI: https://doi.org/10.1145/3329007.3329014.

[57] F. Mannhardt, „Multi-perspective process mining”, dissz., Technische Universiteit Eindhoven, 2018. cím: https://pure.tue.nl/ws/files/90463927/20180207_Mannhardt.pdf.

[58] P. Felli, A. Gianola, M. Montali, A. Rivkin és S. Winkler, „CoCoMoT: conformance checking of multi-perspective processes via SMT”, *Business Process Management: 19th International Conference, BPM 2021, Rome, Italy, September 06–10, 2021, Proceedings*, A. Polyvyanyy, M. T. Wynn, A. Van Looy és M. Reichert, szerk., Lecture Notes in Computer Science sor., Springer, 12875. köt., 2021, 217–234. old. DOI: https://doi.org/10.1007/978-3-030-85469-0_15.

[59] P. Felli, A. Gianola, M. Montali, A. Rivkin és S. Winkler, „Conformance checking with uncertainty via SMT”, *International Conference on Business Process Management*, C. Di Ciccio, R. Dijkman, A. del Río Ortega és S. Rinderle-Ma, szerk., Lecture Notes in Computer Science sor., Springer, Cham, 13420. köt., 2022, 199–216. old. DOI: https://doi.org/10.1007/978-3-031-16103-2_15.

[60] P. Felli, A. Gianola, M. Montali, A. Rivkin és S. Winkler, „A Modular SMT-based Approach for Data-aware Conformance Checking”, *Short Paper Proceedings of the 4th Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis hosted by the 21st International Conference of the Italian Association for Artificial Intelligence (AIxIA 2022), Udine, Italy, November 28, 2022*, L. Geatti, G. Sciavicco és A. Umbrico, szerk., 3311. köt., CEUR-WS.org, Aachen, Germany, 2022, 87–92. old. cím: https://ceur-ws.org/Vol-3311/paper14.pdf.

[61] P. Felli, A. Gianola, M. Montali, A. Rivkin és S. Winkler, „Data-aware conformance checking with SMT”, *Information Systems*, 117. évf., 102230. old., 2023. DOI: https://doi.org/10.1016/j.is.2023.102230.

[62] P. Felli, A. Gianola, M. Montali, A. Rivkin és S. Winkler, „Multi-perspective conformance checking of uncertain process traces: An SMT-based approach”, *Engineering Applications of Artificial Intelligence*, 126. évf., 106895. old., 2023. DOI: https://doi.org/10.1016/j.engappai.2023.106895.

[63] A. Gianola, J. Ko, F. M. Maggi, M. Montali és S. Winkler, „Approximating Multi-perspective Trace Alignment Using Trace Encodings”, *Business Process Management: 21st International Conference, BPM 2023, Utrecht, The Netherlands, September 11–15, 2023, Proceedings*, C. Di Francescomarino, A. Burattin, C. Janiesch és S. Sadiq, szerk., Lecture Notes in Computer Science sor., Springer, Cham, 14159. köt., 2023, 74–91. old. DOI: https://doi.org/10.1007/978-3-031-41620-0_5.

[64] A. Gianola, J. Ko, F. M. Maggi, M. Montali és S. Winkler, „Approximate Conformance Checking: Fast Computation of Multi-Perspective, Probabilistic Alignments”, 2024. DOI: https://dx.doi.org/10.2139/ssrn.4710404.

[65] M. Alizadeh, X. Lu, D. Fahland, N. Zannone és W. M. van der Aalst, „Linking data and process perspectives for conformance analysis”, *Computers & Security*, 73. évf., 172–193. old., 2018. DOI: https://doi.org/10.1016/j.cose.2017.10.010.

[66] A. S. Mozafari Mehr, R. M. de Carvalho és B. van Dongen, „Detecting privacy, data and control-flow deviations in business processes”, *International Conference on Advanced Information Systems Engineering*, Springer, Cham, 2021, 82–91. old. DOI: https://doi.org/10.1007/978-3-030-79108-7_10.

[67] A. S. M. Mehr, R. M. de Carvalho és B. van Dongen, „Identifying the Context of Data Usage to Diagnose Privacy Issues through Process Mining”, *Transactions on Data Privacy*, 16. évf., 2. sz., 123–151. old., 2023. cím: http://www.tdp.cat/issues21/tdp.a456a22.pdf.

[68] A. S. Mozafari Mehr, R. M. de Carvalho és B. van Dongen, „Detecting complex anomalous behaviors in business processes: a multi-perspective conformance checking approach”, *Process Mining Workshops: ICPM 2022 International Workshops, Bozen-Bolzano, Italy, October 23–28, 2022, Revised Selected Papers*, M. Montali, A. Senderovich és M. Weidlich, szerk., Springer, Cham, 468. köt., 2022, 44–56. old. DOI: https://doi.org/10.1007/978-3-031-27815-0_4.

[69] A. S. M. Mehr, R. M. de Carvalho és B. van Dongen, „Explainable conformance checking: Understanding patterns of anomalous behavior”, *Engineering Applications of Artificial Intelligence*, 126. évf., 106827. old., 2023. DOI: https://doi.org/10.1016/j.engappai.2023.106827.

[70] A. S. Mozafari Mehr, R. M. de Carvalho és B. van Dongen, „An association rule mining-based framework for the discovery of anomalous behavioral patterns”, *Advanced Data Mining and Applications: 18th International Conference, ADMA 2022, Brisbane, QLD, Australia, November 28–30, 2022, Proceedings, Part I*, W. Chen, L. Yao, T. Cai, S. Pan, T. Shen és X. Li, szerk., Lecture Notes in Computer Science sor., 13725. köt., Springer, Cham, 2022, 397–412. old. DOI: https://doi.org/10.1007/978-3-031-22064-7_29.

[71] A. S. M. Mehr, R. M. de Carvalho és B. van Dongen, „MLA: A tool for multi-perspective conformance checking of business processes”, *Proceedings of the ICPM Doctoral Consortium and Demo Track 2021 (ICPM-D 2021) co-located with 10th International Conference on Process Mining (ICPM 2021), Eindhoven, The Netherlands, October 30–November 4, 2021*, M. Jans, G. Janssenswillen, A. Kalenkova és F. M. Maggi, szerk., 3098. köt., CEUR-WS.org, Aachen, Germany, 2021, 35–36. old. cím: https://ceur-ws.org/Vol-3098/demo_200.pdf.

[72] A. S. M. Mehr, „Multi-perspective Conformance Checking: Identifying and Understanding Patterns of Anomalous Behavior”, dissz., Technische Universiteit Eindhoven, 2024. cím: https://pure.tue.nl/ws/portalfiles/portal/326564778/20240522_Mozafari_Mehr_hf.pdf.

[73] S. Zhang, L. Genga, H. Yan, X. Lu, H. Duan és U. Kaymak, „Towards multi-perspective conformance checking with fuzzy sets”, *arXiv preprint arXiv:2001.10730*, 2020. DOI: https://doi.org/10.48550/arXiv.2001.10730.

[74] A. Banham, A. H. Ter Hofstede, S. J. J. Leemans, F. Mannhardt, R. Andrews és M. T. Wynn, „Comparing Conformance Checking for Decision Mining: An Axiomatic Approach”, *IEEE Access*, 2024. DOI: https://doi.org/10.1109/ACCESS.2024.3391234.

[75] I. Weber, A. Rogge-Solti, C. Li és J. Mendling, „CCaaS: Online Conformance Checking as a Service.”, *Proceedings of the BPM Demo Session 2015 (BPMD 2015), Innsbruck, Austria, September 2, 2015*, CEUR-WS.org, Aachen, Germany, 2015, 45–49. old. cím: https://ceur-ws.org/Vol-1418/paper10.pdf.

[76] A. Burattin, „Online Conformance Checking for Petri Nets and Event Streams”, *BPM Demo Track and BPM Dissertation Award (BPM-D&DA 2017): Proceedings of the BPM Demo Track and BPM Dissertation Award co-located with 15th International Conference on Business Process Management (BPM 2017), Barcelona, Spain. September 13, 2017*, CEUR-WS.org, Aachen, Germany, 2017. cím: https://ceur-ws.org/Vol-1920/BPM_2017_paper_153.pdf.

[77] B. F. Van Dongen, N. Busi, G. M. Pinna és W. M. van der Aalst, „An iterative algorithm for applying the theory of regions in process mining”, *Proceedings of the workshop on formal approaches to business processes and web services (FABPWS'07)*, Publishing House of University of Podlasie, Siedlce, Poland, 2007, 36–55. old. cím: https://pure.tue.nl/ws/portalfiles/portal/3518540/623172.pdf.

[78] K. Raun, M. Nielsen, A. Burattin és A. Awad, „C-3PA: Streaming Conformance, Confidence and Completeness in Prefix-Alignments”, *Advanced Information Systems Engineering: 35th International Conference, CAiSE 2023, Zaragoza, Spain, June 12–16, 2023, Proceedings*, M. Indulska, I. Reinhartz-Berger, C. Cetina és O. Pastor, szerk., Lecture Notes in Computer Science sor., Springer, Cham, 13901. köt., 2023, 437–453. old. DOI: https://doi.org/10.1007/978-3-031-34560-9_26.

[79] A. Adriansyah, B. F. Van Dongen és N. Zannone, „Controlling break-the-glass through alignment”, *2013 International Conference on Social Computing, Alexandria, VA, USA, September 8–14, 2013*, IEEE, New York, NY, USA, 2013, 606–611. old. DOI: https://doi.org/10.1109/SocialCom.2013.91.

[80] D. Schuster és G. J. Kolhof, „Scalable online conformance checking using incremental prefix-alignment computation”, *Service-Oriented Computing–ICSOC 2020 Workshops: AIOps, CFTIC, STRAPS, AI-PA, AI-IOTS, and Satellite Events, Dubai, United Arab Emirates, December 14–17, 2020, Proceedings*, H. Hacid és tsai., szerk., Lecture Notes in Computer Science sor., Springer, Cham, 12632. köt., 2021, 379–394. old. DOI: https://doi.org/10.1007/978-3-030-76352-7_36.

[81] R. Zaman, M. Hassani és B. F. Van Dongen, „A Framework for Efficient Memory Utilization in Online Conformance Checking”, *arXiv preprint arXiv:2112.13640*, 2021. DOI: https://doi.org/10.48550/arXiv.2112.13640.

[82] R. Zaman, M. Hassani és B. F. Van Dongen, „Efficient memory utilization in conformance checking of process event streams”, *SAC '22: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, April 25–29, 2022*, New York, NY, United States: Association for Computing Machinery, 2022, 437–440. old. DOI: https://doi.org/10.1145/3477314.3507217.

[83] R. Zaman, M. Hassani és B. F. van Dongen, „Conformance checking of process event streams with constraints on data retention”, *Information Systems*, 117. évf., 102228. old., 2023. DOI: https://doi.org/10.1016/j.is.2023.102228.

[84] K. Raun és A. Awad, „I Will Survive: An Online Conformance Checking Algorithm Using Decay Time”, *arXiv preprint arXiv:2211.16702*, 2022. DOI: https://doi.org/10.48550/arXiv.2211.16702.

[85] K. Raun, R. Tommasini és A. Awad, „I Will Survive: An Event-driven Conformance Checking Approach Over Process Streams”, *DEBS '23: Proceedings of the 17th ACM International Conference on Distributed and Event-based Systems, Neuchatel, Switzerland, June 27–30, 2023*, 2023, 49–60. old. DOI: https://doi.org/10.1145/3583678.3596887.

[86] R. Zaman, M. Hassani és B. F. Van Dongen, „Prefix imputation of orphan events in event stream processing”, *Frontiers in big Data*, 4. évf., 2021. DOI: https://doi.org/10.3389/fdata.2021.705243.

[87] A. Burattin, „Online Soft Conformance Checking: Any Perspective Can Indicate Deviations”, *arXiv preprint arXiv:2201.09222*, 2022. DOI: https://doi.org/10.48550/arXiv.2201.09222.

[88] F. Stertz, J. Mangler és S. Rinderle-Ma, „Data-driven improvement of online conformance checking”, *2020 IEEE 24th international enterprise distributed object computing conference (EDOC), Eindhoven, Netherlands, October 5–8, 2020*, IEEE, New York, NY, USA, 2020, 187–196. old. DOI: https://doi.org/10.1109/EDOC49727.2020.00031.

[89] U. Celik és E. Akçetin, „Process mining tools comparison”, *Online Academic Journal of Information Technology*, 9. évf., 34. sz., 97–104. old., 2018. DOI: https://doi.org/10.5824/1309-1581.2018.4.007.x.

[90] P. Drakoulogkonas és D. Apostolou, „On the selection of process mining tools”, *Electronics*, 10. évf., 4. sz., 451. old., 2021. DOI: https://doi.org/10.3390/electronics10040451.

[91] D. Viner, M. Stierle és M. Matzner, „A process mining software comparison”, *arXiv preprint arXiv:2007.14038*, 2020. DOI: https://doi.org/10.48550/arXiv.2007.14038.

[92] A. Berti, S. van Zelst és D. Schuster, „PM4Py: A process mining library for Python”, *Software Impacts*, 17. évf., 100556. old., 2023. DOI: https://doi.org/10.1016/j.simpa.2023.100556.

[93] A. Kaouni, G. Theodoropoulou, A. Bousdekis, A. Voulodimos és G. Miaoulis, „Visual analytics in process mining for supporting business process improvement”, *Proceedings of the 1st International Conference on Novelties in Intelligent Digital Systems (NIDS 2021), Athens, Greece, September 30–October 1, 2021*, C. Frasson, K. Kabassi és A. Voulodimos, szerk., Frontiers in Artificial Intelligence and Applications sor., 338. köt., IOS Press, Amsterdam, The Netherlands, 2021, 166–175. old. DOI: https://doi.org/10.3233/FAIA210089.

[94] A. Berti, C.-Y. Li, D. Schuster és S. J. van Zelst, „The process mining toolkit (PMTK): Enabling advanced process mining in an integrated fashion”, M. Jans, G. Janssenswillen, A. Kalenkova és F. M. Maggi, szerk., 3098. köt., CEUR-WS.org, Aachen, Germany, 2021, 43–44. old. cím: https://ceur-ws.org/Vol-3098/demo_206.pdf.

[95] M. Song és W. M. van der Aalst, „Supporting process mining by showing events at a glance”, *Proceedings of the 17th Annual Workshop on Information Technologies and Systems (WITS)*, 2007, 139–145. old. cím: https://vdaalst.com/publications/p433.pdf.

[96] I. Merkoureas, A. Kaouni, G. Theodoropoulou, A. Bousdekis, A. Voulodimos és G. Miaoulis, „Smyrida: A web application for process mining and interactive visualization”, *SoftwareX*, 22. évf., 101327. old., 2023. DOI: https://doi.org/10.1016/j.softx.2023.101327.

[97] D. Schuster, S. J. van Zelst és W. M. van der Aalst, „Cortado: A dedicated process mining tool for interactive process discovery”, *SoftwareX*, 22. évf., 101373. old., 2023. DOI: https://doi.org/10.1016/j.softx.2023.101373.

[98] A. Yeshchenko és J. Mendling, „A survey of approaches for event sequence analysis and visualization using the esevis framework”, *arXiv preprint arXiv:2202.07941*, 2022. DOI: https://doi.org/10.48550/arXiv.2202.07941.

[99] P. T. Hornix, „Performance analysis of business processes through process mining”, 2007. cím: https://pure.tue.nl/ws/portalfiles/portal/47012708/625917-1.pdf.

[100] J.-R. Rehse, L. Pufahl, M. Grohs és L.-M. Klein, „Process mining meets visual analytics: the case of conformance checking”, *arXiv preprint arXiv:2209.09712*, 2022. DOI: https://doi.org/10.48550/arXiv.2209.09712.

[101] F. Mannhardt, M. De Leoni és H. A. Reijers, „The multi-perspective process explorer”, *Proceedings of the Demo Session of the 13th International Conference on Business Process Management (BPM 2015), Innsbruck, Austria, August 31–September 3, 2015*, F. Daniel és S. Zugal, szerk., 1418. köt., CEUR-WS.org, Aachen, Germany, 2015, 130–134. old. cím: https://pure.tue.nl/ws/portalfiles/portal/36470763/mannmult2015.pdf.

[102] B. F. Hompes, J. C. Buijs és W. M. van der Aalst, „A generic framework for context-aware process performance analysis”, *On the Move to Meaningful Internet Systems: OTM 2016 Conferences: Confederated International Conferences: CoopIS, C&TC, and ODBASE 2016, Rhodes, Greece, October 24-28, 2016, Proceedings*, C. Debruyne és tsai., szerk., Lecture Notes in Computer Science sor., Springer, Cham, 10033. köt., 2016, 300–317. old. DOI: https://doi.org/10.1007/978-3-319-48472-3_17.

[103] N. Sidorova, C. Stahl és N. Trčka, „Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible”, *Information Systems*, 36. évf., 7. sz., 1026–1043. old., 2011, Special Issue: Advanced Information Systems Engineering (CAiSE'10). DOI: https://doi.org/10.1016/j.is.2011.04.004.

[104] M. De Leoni és W. M. Van Der Aalst, „Data-aware process mining: discovering decisions in processes using alignments”, *SAC '13: Proceedings of the 28th annual ACM symposium on applied computing, Coimbra, Portugal, March 18–22, 2013*, ACM, New York, NY, USA, 2013, 1454–1461. old. DOI: https://doi.org/10.1145/2480362.2480633.

[105] A. Adriansyah, „Aligning observed and modeled behavior”, dissz., Technische Universiteit Eindhoven, 2014. DOI: https://doi.org/10.6100/IR770080.

[106] S. J. van Zelst, A. Bolt és B. F. van Dongen, „Computing alignments of event data and process models”, *Transactions on Petri Nets and Other Models of Concurrency XIII*, M. Koutny, L. M. Kristensen és W. Penczek, szerk., Springer, Berlin, Heidelberg, 2018, 1–26. old. DOI: https://doi.org/10.1007/978-3-662-58381-4_1.

[107] F. Taymouri, „Light methods for conformance checking of business processes”, dissz., Universitat Politècnica de Catalunya, 2018. cím: http://hdl.handle.net/2117/127494.

[108] A. Schrijver, *Theory of linear and integer programming*. John Wiley & Sons, New York, NY, USA, 1998.

[109] S. P. Bradley, A. C. Hax és T. L. Magnanti, *Applied mathematical programming*. Addison-Wesley, Boston, MA, USA, 1977.

[110] P. E. Hart, N. J. Nilsson és B. Raphael, „A formal basis for the heuristic determination of minimum cost paths”, *IEEE transactions on Systems Science and Cybernetics*, 4. évf., 2. sz., 100–107. old., 1968. DOI: https://doi.org/10.1109/TSSC.1968.300136.

[111] V. Bloemen, J. van de Pol és W. M. van der Aalst, „Symbolically aligning observed and modelled behaviour”, *2018 18th International Conference on Application of Concurrency to System Design (ACSD), Bratislava, Slovakia, June 25–29, 2018*, IEEE, New York, NY, USA, 2018, 50–59. old. DOI: https://doi.org/10.1109/ACSD.2018.00008.

[112] T. Munzner, *Visualization analysis and design*. CRC press, 2014, 1. köt.

[113] Y. Han, A. Rozga, N. Dimitrova, G. D. Abowd és J. Stasko, „Visual analysis of proximal temporal relationships of social and communicative behaviors”, *Computer Graphics Forum*, 34. évf., 3. sz., 51–60. old., 2015. DOI: https://doi.org/10.1111/cgf.12617.

[114] *ProM*, 6.9. verzió, Process Mining Group és TU/e. cím: http://www.promtools.org/ (elérés dátuma 2023. 03. 18.).

[115] A. Berti, S. J. Van Zelst és W. van der Aalst, *PM4Py*, 2.7.5.2. verzió, Fraunhofer Institute for Applied Information Technology (FIT). cím: https://pm4py.fit.fraunhofer.de/ (elérés dátuma 2023. 03. 18.).

[116] L. Perron és V. Furnon, *OR-Tools*, v9.6. verzió, Google, 2023. márc. 13. cím: https://developers.google.com/optimization/.

[117] D. Schuster és S. J. van Zelst, *Online Process Monitoring Using Incremental State-Space Expansion: An Exact Algorithm*. cím: https://github.com/fit-daniel-schuster/online_process_monitoring_using_incremental_state-space_expansion_an_exact_algorithm (elérés dátuma 2023. 03. 18.).

[118] B. L. Golden, J. S. DeArmon és E. K. Baker, „Computational experiments with algorithms for a class of routing problems”, *Computers & Operations Research*, 10. évf., 1. sz., 47–59. old., 1983. DOI: https://doi.org/10.1016/0305-0548(83)90026-6.

[119] L. Chapleau, J. A. Ferland, G. Lapalme és J.-M. Rousseau, „A parallel insert method for the capacitated arc routing problem”, *Operations Research Letters*, 3. évf., 2. sz., 95–99. old., 1984. DOI: https://doi.org/10.1016/0167-6377(84)90049-X.

[120] G. Ulusoy, „The fleet size and mix problem for capacitated arc routing”, *European Journal of Operational Research*, 22. évf., 3. sz., 329–337. old., 1985. DOI: https://doi.org/10.1016/0377-2217(85)90252-8.

[121] W. L. Pearn, „Augment-insert algorithms for the capacitated arc routing problem”, *Computers & Operations Research*, 18. évf., 2. sz., 189–198. old., 1991. DOI: https://doi.org/10.1016/0305-0548(91)90089-A.

[122] L. Santos, J. Coutinho-Rodrigues és J. R. Current, „An improved heuristic for the capacitated arc routing problem”, *Computers & Operations Research*, 36. évf., 9. sz., 2632–2637. old., 2009. DOI: https://doi.org/10.1016/j.cor.2008.11.005.

[123] R. K. Arakaki és F. L. Usberti, „An efficiency-based path-scanning heuristic for the capacitated arc routing problem”, *Computers & Operations Research*, 103. évf., 288–295. old., 2019. DOI: https://doi.org/10.1016/j.cor.2018.11.018.

[124] P. Beullens, L. Muyldermans, D. Cattrysse és D. Van Oudheusden, „A guided local search heuristic for the capacitated arc routing problem”, *European Journal of Operational Research*, 147. évf., 3. sz., 629–643. old., 2003. DOI: https://doi.org/10.1016/S0377-2217(02)00334-X.

[125] A. Hertz, G. Laporte és M. Mittaz, „A tabu search heuristic for the capacitated arc routing problem”, *Operations research*, 48. évf., 1. sz., 129–135. old., 2000. DOI: https://doi.org/10.1287/opre.48.1.129.12455.

[126] J. Brandão és R. Eglese, „A deterministic tabu search algorithm for the capacitated arc routing problem”, *Computers & Operations Research*, 35. évf., 4. sz., 1112–1126. old., 2008. DOI: https://doi.org/10.1016/j.cor.2006.07.007.

[127] M. Polacek, K. F. Doerner, R. F. Hartl és V. Maniezzo, „A variable neighborhood search for the capacitated arc routing problem with intermediate facilities”, *Journal of Heuristics*, 14. évf., 405–423. old., 2008. DOI: https://doi.org/10.1007/s10732-007-9050-2.

[128] F. L. Usberti, P. M. França és A. L. M. França, „GRASP with evolutionary path-relinking for the capacitated arc routing problem”, *Computers & Operations Research*, 40. évf., 12. sz., 3206–3217. old., 2013. DOI: https://doi.org/10.1016/j.cor.2011.10.014.

[129] Y. Mei, K. Tang és X. Yao, „A global repair operator for capacitated arc routing problem”, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39. évf., 3. sz., 723–734. old., 2009. DOI: https://doi.org/10.1109/TSMCB.2008.2008906.

[130] P. Lacomme, C. Prins és W. Ramdane-Chérif, „A genetic algorithm for the capacitated arc routing problem and its extensions”, *Workshops on Applications of Evolutionary Computing (EvoWorkshops 2001): EvoCOP, EvoFlight, EvoIASP, EvoLearn, and EvoSTIM Como, Italy, April 18–20, 2001, Proceedings*, Springer, Berlin, Heidelberg, Germany, 2001, 473–483. old. DOI: https://doi.org/10.1007/3-540-45365-2_49.

[131] P. Lacomme, C. Prins és W. Ramdane-Cherif, „Competitive memetic algorithms for arc routing problems”, *Annals of Operations Research*, 131. évf., 159–185. old., 2004. DOI: https://doi.org/10.1023/B:ANOR.0000039517.35989.6d.

[132] P. Lacomme, C. Prins és A. Tanguy, „First competitive ant colony scheme for the CARP”, *4th International Workshop on Ant Colony Optimization and Swarm Intelligence, ANTS 2004, Brussels, Belgium, September 5–8, 2004, Proceedings*, Springer, Berlin, Heidelberg, Germany, 2004, 426–427. old. DOI: https://doi.org/10.1007/978-3-540-28646-2_48.

[133] L. Santos, J. Coutinho-Rodrigues és J. R. Current, „An improved ant colony optimization based algorithm for the capacitated arc routing problem”, *Transportation Research Part B: Methodological*, 44. évf., 2. sz., 246–266. old., 2010. DOI: https://doi.org/10.1016/j.trb.2009.07.004.

[134] M. Tagmouti, M. Gendreau és J.-Y. Potvin, „A dynamic capacitated arc routing problem with time-dependent service costs”, *Transportation Research Part C: Emerging Technologies*, 19. évf., 1. sz., 20–28. old., 2011. DOI: https://doi.org/10.1016/j.trc.2010.02.003.

[135] C. Archetti, G. Guastaroba és M. G. Speranza, „Reoptimizing the rural postman problem”, *Computers & operations research*, 40. évf., 5. sz., 1306–1313. old., 2013. DOI: https://doi.org/10.1016/j.cor.2012.12.010.

[136] Y. Mei, K. Tang és X. Yao, „Evolutionary computation for dynamic capacitated arc routing problem”, *Evolutionary Computation for Dynamic Optimization Problems*, Springer, 2013, 377–401. old. DOI: https://doi.org/10.1007/978-3-642-38416-5_15.

[137] A. Yazici, G. Kirlik, O. Parlaktuna és A. Sipahioglu, „A dynamic path planning approach for multirobot sensor-based coverage considering energy constraints”, *IEEE transactions on cybernetics*, 44. évf., 3. sz., 305–314. old., 2013. DOI: https://doi.org/10.1109/TCYB.2013.2253605.

[138] M. Liu, H. K. Singh és T. Ray, „A memetic algorithm with a new split scheme for solving dynamic capacitated arc routing problems”, *2014 IEEE Congress on Evolutionary Computation (CEC), Beijing, China, July 6–11, 2014*, IEEE, 2014, 595–602. old. DOI: https://doi.org/10.1109/CEC.2014.6900358.

[139] M. Monroy-Licht, C. A. Amaya, A. Langevin és L.-M. Rousseau, „The rescheduling arc routing problem”, *International Transactions in Operational Research*, 24. évf., 6. sz., 1325–1346. old., 2017. DOI: https://doi.org/10.1111/itor.12346.

[140] W. Padungwech, „Heuristic algorithms for dynamic capacitated arc routing”, dissz., School of Mathematics, Cardiff University, 2018. cím: https://orca.cardiff.ac.uk/id/eprint/111021/.

[141] W. Padungwech, J. Thompson és R. Lewis, „Effects of update frequencies in a dynamic capacitated arc routing problem”, *Networks*, 76. évf., 4. sz., 522–538. old., 2020. DOI: https://doi.org/10.1002/net.21990.

[142] D. Karaboga és B. Gorkemli, „A quick artificial bee colony-qABC-algorithm for optimization problems”, *2012 International symposium on innovations in intelligent systems and applications, Trabzon, Turkey, July 2–4, 2012*, IEEE, 2012, 1–5. old. DOI: https://doi.org/10.1109/INISTA.2012.6247010.

[143] M. Albayrak és N. Allahverdi, „Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms”, *Expert Systems with Applications*, 38. évf., 3. sz., 1313–1320. old., 2011. DOI: https://doi.org/10.1016/j.eswa.2010.07.006.

[144] A. S. Bhagade és P. V. Puranik, „Artificial bee colony (ABC) algorithm for vehicle routing optimization problem”, *International Journal of Soft Computing and Engineering (IJSCE)*, 2. évf., 2. sz., 329–333. old., 2012.

[145] P. Consoli és X. Yao, „Diversity-driven selection of multiple crossover operators for the capacitated arc routing problem”, *Evolutionary Computation in Combinatorial Optimisation: 14th European Conference, EvoCOP 2014, Granada, Spain, April 23–25, 2014, Revised Selected Papers*, Springer, Berlin, Heidelberg, 2014, 97–108. old. DOI: https://doi.org/10.1007/978-3-662-44320-0_9.

[146] P. A. Lopez és tsai., „Microscopic traffic simulation using sumo”, *2018 21st international conference on intelligent transportation systems (ITSC), Maui, HI, USA, November 4–7, 2018*, IEEE, 2018, 2575–2582. old. DOI: https://doi.org/10.1109/ITSC.2018.8569938.

[147] H. Tong, L. L. Minku, S. Menzel, B. Sendhoff és X. Yao, „Benchmarking Dynamic Capacitated Arc Routing Algorithms Using Real-World Traffic Simulation”, *2022 IEEE Congress on Evolutionary Computation (CEC), Padua, Italy, July 18–23, 2022*, IEEE, 2022, 1–8. old. DOI: https://doi.org/10.1109/CEC55065.2022.9870399.

[148] H. Tong, L. L. Minku, S. Menzel, B. Sendhoff és X. Yao, „A hybrid local search framework for the dynamic capacitated arc routing problem”, *GECCO '21: Proceedings of the Genetic and Evolutionary Computation Conference Companion, Lille, France, July 10–14, 2021*, ACM, New York, NY, USA, 2021, 139–140. old. DOI: https://doi.org/10.1145/3449726.3459450.

[149] M. de Leoni és F. Mannhardt, „Road Traffic Fine Management Process”, 2015. DOI: https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.

[150] F. Mannhardt, „Hospital Billing - Event Log”, 2017. DOI: https://doi.org/10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741.

[151] E. J. Willemse és J. W. Joubert, „Splitting procedures for the mixed capacitated arc routing problem under time restrictions with intermediate facilities”, *Operations Research Letters*, 44. évf., 5. sz., 569–574. old., 2016. DOI: https://doi.org/10.1016/j.orl.2016.06.001.

[152] R. Durstenfeld, „Algorithm 235: random permutation”, *Communications of the ACM*, 7. évf., 7. sz., 420. old., 1964. DOI: https://doi.org/10.1016/S0377-2217(02)00334-X.

[153] M. Kiuchi, Y. Shinano, R. Hirabayashi és Y. Saruwatari, „An exact algorithm for the capacitated arc routing problem using parallel branch and bound method”, *Spring national conference of the operational research society of Japan*, INFORMS Catonsville, MD, USA, 1995, 28–29. old.

[154] J. S. DeArmon, „A comparison of heuristics for the capacitated Chinese postman problem”, dissz., University of Maryland, 1981.

[155] E. Benavent, V. Campos, A. Corberán és E. Mota, „The capacitated arc routing problem: lower bounds”, *Networks*, 22. évf., 7. sz., 669–690. old., 1992. DOI: https://doi.org/10.1002/net.3230220706.

[156] M. Guériau és I. Dusparic, „Quantifying the impact of connected and autonomous vehicles on traffic efficiency and safety in mixed traffic”, *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), Rhodes, Greece, September 20–23, 2020*, IEEE, 2020, 1–8. old. DOI: https://doi.org/10.1109/ITSC45102.2020.9294174.

Függelékek

A. függelék

Fordítások

Több perspektívás online megfelelőség-ellenőrzés

Angol megnevezés	**Magyar fordítás**
action-oriented process mining	cselekvésorientált folyamatbányászat
activity cost function	tevékenységköltség-függvény
activity instance	tevékenységpéldány
activity label function	tevékenységcímke függvény
alignment	igazítás
bottleneck	szűk keresztmetszet
case identifier	esetazonosító
complex event stream	összetett eseményfolyam
Computing Conformance Modulo Theories	megfelelőség kiszámítási moduló elméletek
conformance checking	megfelelőség ellenőrzés
control-flow	vezérlésfolyam
Data Petri Net	adat Petri-háló
(data-dependent) guard	(adatfüggő) őr
enhancement	bővítés
event	esemény
event attribute	esemény attribútum
event log	eseménynapló
event stream	eseményfolyam
firing sequence	tüzelési szekvencia
guard expression	őrkifejezés
guard-precision	őr-precízség
guard-recall	őr-visszaidézés
Integer Linear Programming	egészértékű lineáris programozás

invisible transition	láthatatlan átmenet
log step	eseménynapló lépés
Mixed-Integer Linear Programming	vegyes egészértékű lineáris programozás
model step	modell lépés
Optimal Variable Assignment	optimális változó hozzárendelés
Optimal Variable Assignment Problem	optimális változó hozzárendelési probléma
performance analysis	teljesítményelemzés
predictive process mining	prediktív folyamatbányászat
prefix-alignment	előtag-igazítás
prime variable	prímváltozó
process discovery	folyamat felfedezés
process mining	folyamatbányászat
(process) variable	(folyamat)változó
rule checking	szabályellenőrzés
Satisfiability Modulo Theories	kielégíthetőségi modulelméletek
single event stream	egyszerű eseményfolyam
Synchronous Product Net	szinkron szorzat háló
token replay	token visszajátszás
trace	nyomvonal
trace net	nyomvonalháló
trace variant	nyomvonalváltozat
variable cost function	változóköltség-függvény
variable label function	változócímke függvény
workflow net	munkafolyamat-háló
write operation	írási művelet

Statikus és dinamikus kapacitáskorlátos ív útvonal-tervezési probléma

Angol megnevezés	**Magyar fordítás**
2-opt	kétopciós
Ant Colony Optimization Algorithm with Path Relinking	hangya kolónia optimalizációs algoritmus útvonal-újrakapcsolással
augment-insert method	bővítési-beillesztési módszer
augment-merge	bővítő-egyesítő
Directed Acyclic Graph	irányított aciklikus gráf
employed bee	dolgozó méh

Greedy Sub Tour Mutation	mohó résztúra mutáció
Hybrid Local Search (algorithm)	hibrid lokális kereső (algoritmus)
Hybrid Metaheuristic Approach	hibrid metaheurisztikus megközelítés
inversion	inverzió
Memetic Algorithm with Extended Neighborhood Search	memetikus algoritmus kiterjesztett szomszédsági kereséssel
merge-split	egyesítés-szétválasztás
(move) operator	(mozgási) művelet
onlooker bee	megfigyelő méh
parallel-insert method	párhuzamos beillesztési módszer
path-scanning	útkereső
Random Solution Generation	véletlenszerű megoldás generátor
Randomized Path-Scanning Heuristic	véletlenszerű útkeresési heurisztika
Repair-based Tabu Search	javításalapú tabu keresés
scout bee	felderítő méh
service log	szolgáltatási napló
(single) insertion	(egyszeri) betoldás
swap	csere
Tabu Search Algorithm	tabu kereső algoritmus
Traveling Salesman Problem	utazó ügynök probléma
Ulusoy's tour splitting method	Ulusoy-féle túrafelosztási módszer
Vehicle Routing Problem	jármű útválasztási probléma

B. függelék

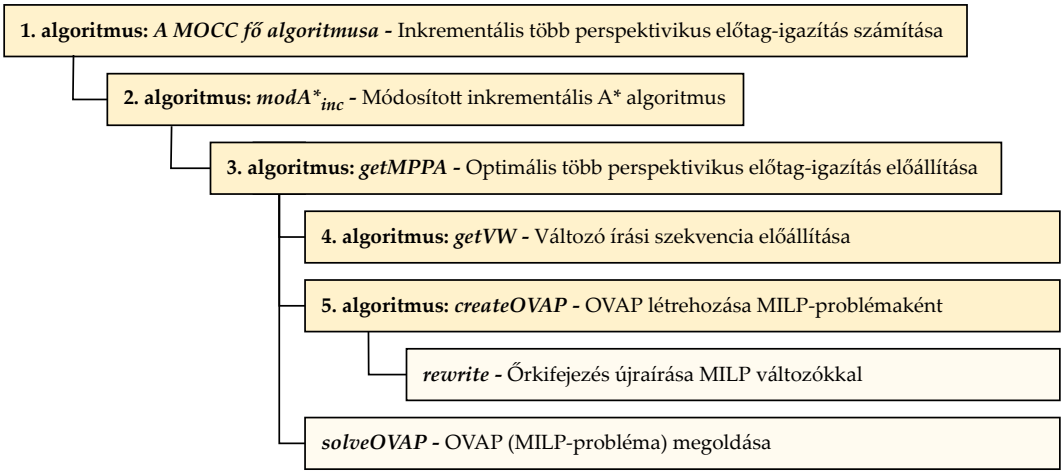
Algoritmusok kapcsolata

Ebben a függelékben a dolgozatban bemutatott megoldások által használt algoritmusok vannak rendszerezve az átláthatóság növelése érdekében.

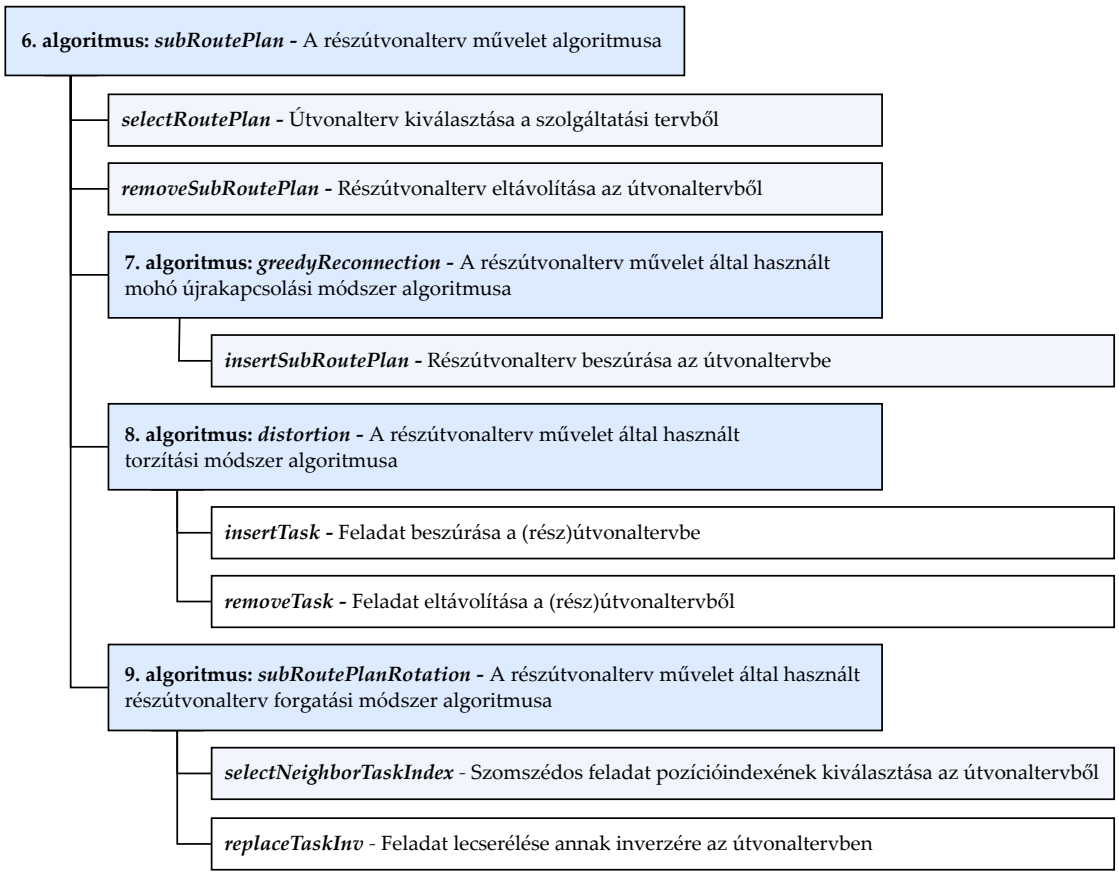
Az alábbi ábrákon (B.1–B.5. ábrák) látható, hogy egy-egy megoldás esetében milyen hierarchikus kapcsolatban állnak egymással az algoritmusok. Az ábrák tetején a fő algoritmus van. Az általa használt algoritmusok hozzá kötve, alatta helyezkednek el. A szín a megoldásra utal, a szín erőssége pedig további jelentést hordoz. A bemutatott, pszeudokóddal rendelkező algoritmusok élénk színnel, a csak szöveges leírással bemutatott vagy csak megemlített algoritmusok pedig halvány színnel vannak jelölve. A fehér szín azt jelzi, hogy nem kötődik szorosan az adott megoldáshoz, tehát az más megoldásokban is alkalmazható.

A bemutatott megoldások, valamint a hozzájuk tartozó ábrák, algoritmusok és színek a következők:

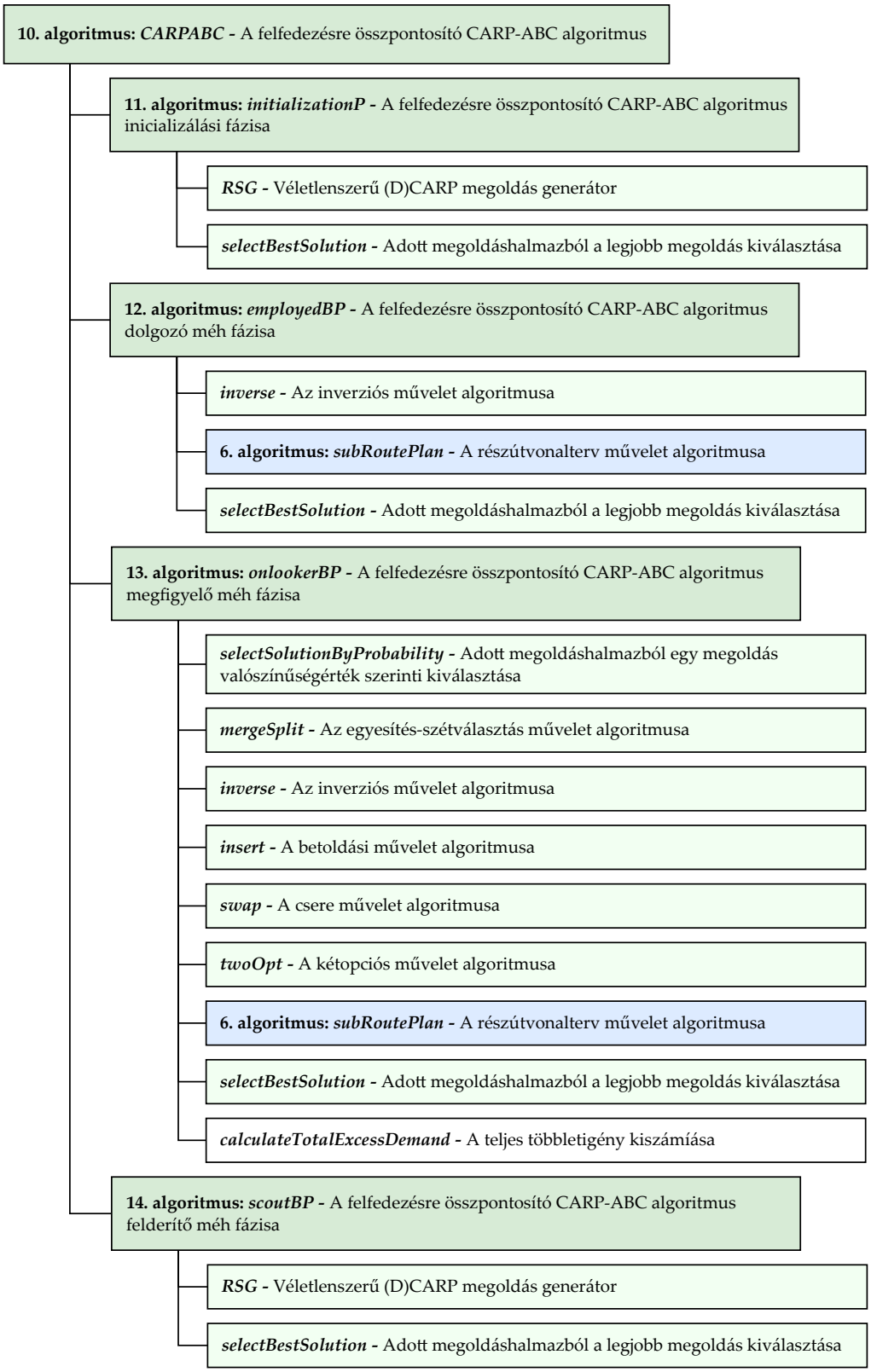
- A javasolt MOCC módszer (2.3. szakasz): B.1. ábra (1–5. algoritmusok, sárga)
- A részútvonalterv művelet (3.3. szakasz): B.2. ábra (6–9. algoritmusok, kék)
- A javasolt CARP-ABC algoritmusok (3.4. szakasz)
  - felfedezésre összpontosító változat: B.3. ábra (10–14. algoritmusok, zöld)
  - feltárásra összpontosító változat: B.4. ábra (15. algoritmus, zöld)
- A javasolt adat-vezérelt DCARP keretrendszer (3.5. szakasz): B.5. ábra (16–24. algoritmusok, piros)



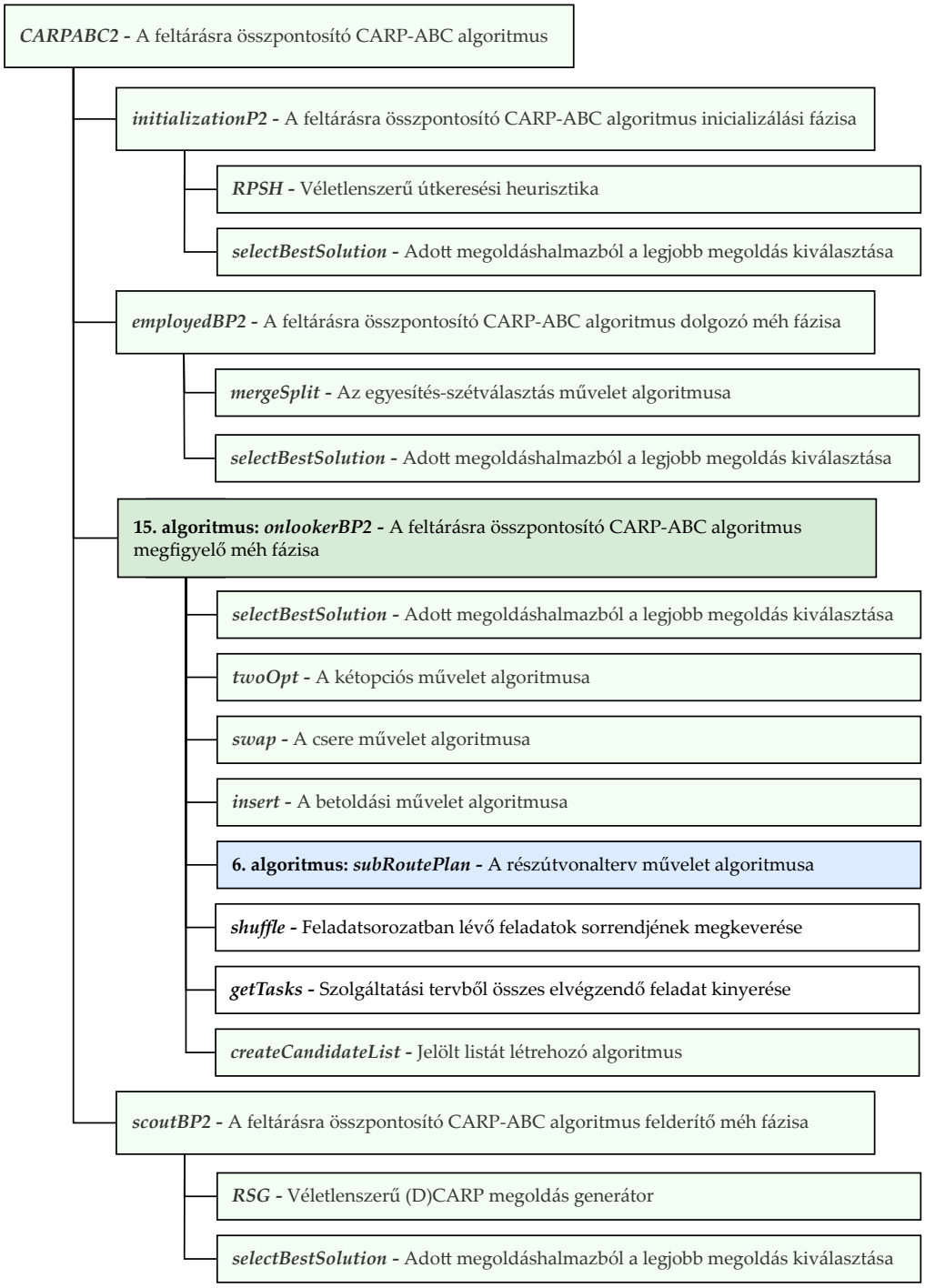
B.1. ábra. A javasolt MOCC módszer által használt algoritmusok kapcsolata



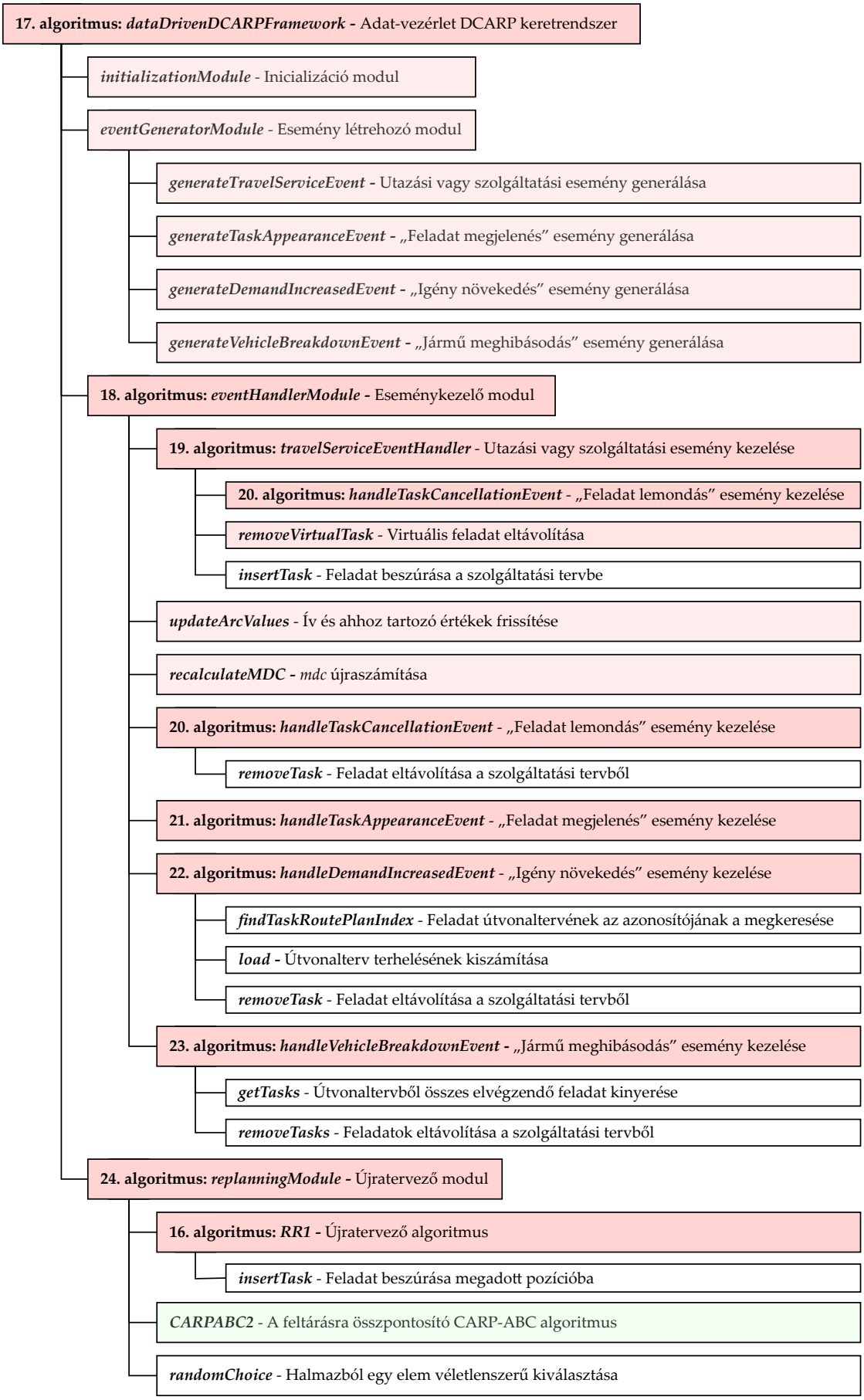
B.2. ábra. A részútvonalterv művelet által használt algoritmusok kapcsolata



B.3. ábra. A felfedezésre összpontosító CARP-ABC algoritmus által használt algoritmusok kapcsolata



B.4. ábra. A feltárásra összpontosító CARP-ABC algoritmus által használt algoritmusok kapcsolata



B.5. ábra. A javasolt adat-vezérelt DCARP keretrendszer által használt algoritmusok kapcsolata

C. függelék

A „Több perspektívás online megfelelőség-ellenőrzés” fejezethez tartozó függelékek

C.1. A felhasznált folyamatok

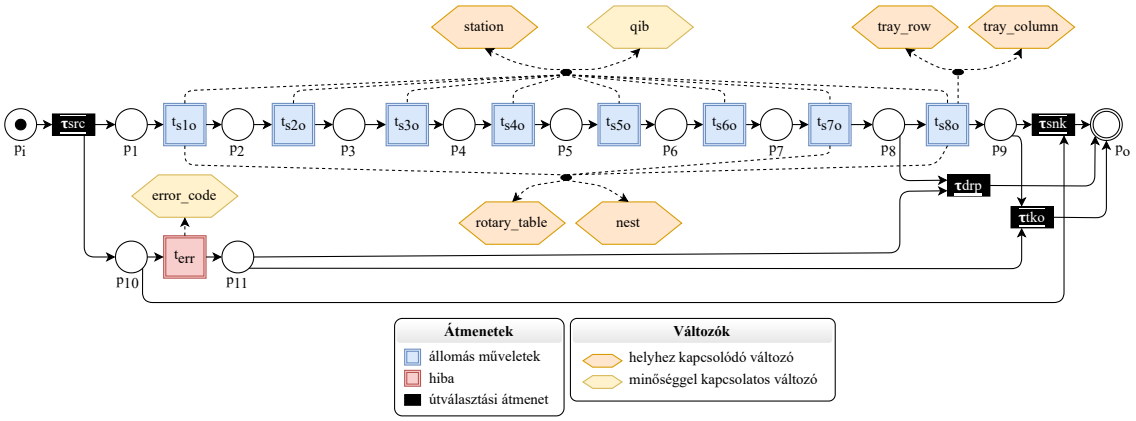
C.1.1. Automatizált tekercs összeszerelési folyamat

A folyamatot két gépsor valósítja meg: egy tekercsgyártó és egy összeszerelő gépsor. Az összeszerelő gépsornak összesen 8 állomása van és a 4. állomáson kapcsolódik hozzá a tekercsgyártó gépsor. A munkámban csak az összeszerelési folyamatot vizsgáltam, mivel az tekinthető a fő folyamatnak. A tekercsgyártási folyamat egy mellékfolyamat, aminek a kimenete az összeszerelési folyamat egyik bemenete.

Az automatizált tekercs összeszerelési folyamat DPN folyamatmodellje a C.1. ábrán látható. Az átmenetekhez rendelt őrkifejezéseket a C.1. táblázat tartalmazza. A színes átmenetek megfigyelhető tevékenységeknek felelnek meg, míg a fekete átmenetekhez nem tartoznak megfigyelhető tevékenységek, azok útválasztási célokat szolgálnak. A kék színű átmenetek állomás műveleteket jelölnek, a piros színű átmenet pedig hiba bekövetkezést jelöl. Minden változó az adott tevékenységgel kapcsolatos esemény egy attribútumát jelöli. A változók tartalom alapján két fő csoportba oszthatók: helyhez kapcsolódó változók és minőséggel kapcsolatos változók. A helyhez kapcsolódó változók a termék fizikai helyéről, a minőséggel kapcsolatos változók pedig a termék (vélt) minőségéről tartalmaznak adatot. A nyilak a változó írási műveleteket jelölik. Látható, hogy csak a vezérlésfolyam perspektívából nézve egyszerű, de a többi perspektívát is figyelembe véve már összetett folyamatról van szó. A folyamat összesen 7 folyamatváltozót, 25 változóírási műveletet és 12 komplex őrkifejezést alkalmaz.

Az automatizált összeszerelési folyamat 8 egymást követő állomásműveletből áll. Ezeket a „t_{s1o}”, „t_{s2o}”, ..., „t_{s8o}” címkével jelölt átmenetek képviselik. A hiba bekövetkezés, amit a „t_{err}” címkéjű átmenet jelöl, bármikor előfordulhat a folyamat végrehajtása során. Ha hiba lép fel, az irányítórendszer egy háromjegyű hibakódot rögzít a termékhez a naplófájlba, amely meghatározza mind az állomást, mind a gép által azonosított konkrét problématípust. Például, a „701” hibakód 1-es típusú hibát jelent a 7. állomáson. Miután egy terméket a rendszer hibásnak minősített, további műveleteket már nem végez rajta. A minőségjelző bit, rövidítve „qib”, azt tárolja,

hogy az állomáson végzett műveletek sikeresek („1” érték esetén) vagy sikertelenek („0” érték esetén) voltak.



C.1. ábra. Az automatizált tekercs összeszerelési folyamat DPN folyamatmodellje

C.1. táblázat. Az automatizált tekercs összeszerelési folyamat DPN folyamatmodelljében használt őrkifejezések

Átmenet	**Őrkifejezés**
t_{s1o}	$station' = 1 \wedge rotary\_table' =$“M”$\wedge nest' \geq 1 \wedge nest' \leq 8$
t_{s2o}, t_{s3o}, ..., t_{s6o}	$station' = station + 1 \wedge ((qib = 1 \wedge error\_code = 0) \vee (qib = 0 \wedge qib' = 0 \wedge error\_code \neq 0))$
t_{s7o}	$station' = 7 \wedge ((qib = 1 \wedge rotary\_table' =$“S”$\wedge nest' \geq 1 \wedge nest' \leq 4) \vee (qib = 0 \wedge rotary\_table =$“M”$\wedge nest' \geq 1 \wedge nest' \leq 8)) \wedge ((qib = 1 \wedge error\_code = 0) \vee (qib = 0 \wedge qib' = 0 \wedge error\_code \neq 0))$
t_{s8o}	$station' = 8 \wedge (rotary\_table' =$“M”$\wedge nest' \geq 1 \wedge nest' \leq 8) \wedge (tray\_row' > 0 \wedge tray\_column' > 0) \wedge ((qib = 1 \wedge error\_code = 0) \vee (qib = 0 \wedge qib' = 0 \wedge error\_code > 700))$
t_{err}	$error\_code' \geq station * 100 + 1 \wedge error\_code' < station * 100 + 100 \wedge qib = 0$
t_{drp}	$station = 7 \wedge qib = 0 \wedge error\_code < 700$
t_{tko}	$station = 8 \wedge qib = 0 \wedge error\_code > 700$
t_{snk}	$station = 8 \wedge qib = 1 \wedge error\_code = 0$

A termékek összeszerelése párhuzamosan történik forgóasztalokon. Az „M” jelzésű fő forgóasztal 8 fészekkel (azaz terméktartóval) rendelkezik, míg a 7. állomáson lévő „S” jelzésű kisebb forgóasztal 4 fészekkel. Az utolsó állomáson a jó termékek és a 700 feletti hibakóddal rendelkező hibás termékek külön tálcákra kerülnek elhelyezésre. Az összes többi hibás termék a 7. állomáson egy selejtgyűjtő dobozba kerül eldobásra. A „tray_row” és a „tray_column” attribútumok tárolják a termék pozícióját a tálcán, az az, hogy melyik sorba és oszlopba lett a termék elhelyezve.

Az eredeti eseménynaplókban minden egyes sorban egy teljes nyomvonal van (tehát 1 sor = 1 nyomvonal). Minden egyes állomásműveletthez egy kezdő időpont

és egy ezredmásodpercben mért időtartam érték van rögzítve. Mivel az attribútumértékek csak az állomásműveletek végrehajtásának befejezése után íródnak a naplófájlba, a naplófájlokat összetett eseményfolyam formátumúra alakítottam át. A C.2. táblázat az eredményül kapott fájlokban található attribútumokat mutatja be, az értéktípus és (ahol releváns) az értéktartomány vagy formátum megjelölésével. Az attribútumok táblázatban lévő sorrendje megegyezik azoknak a fájlokban szereplő sorrendjével.

C.2. táblázat. Az automatizált tekercs összeszerelési folyamat eseménynaplóiban használt attribútumok leírása

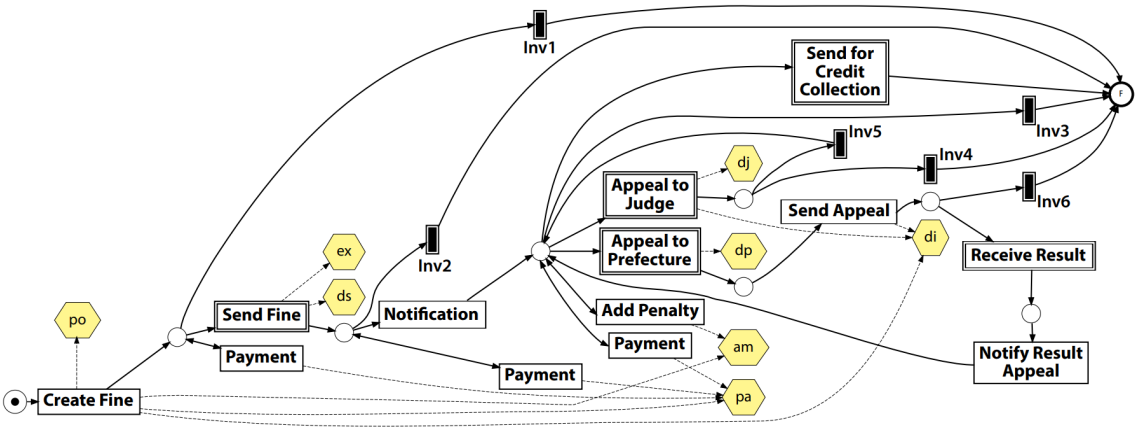
Attribútum neve	**Értéktípus**	**Értéktartomány vagy formátum**
case_id	egész szám	
activity	karakterlánc	[“t_{s1o}”, “t_{s2o}”, …, “t_{s8o}”, “t_{err}”]
start_timestamp	dátum idő	%Y-%m-%d %H:%M:%S.%f
complete_timestamp	dátum idő	%Y-%m-%d %H:%M:%S.%f
station	egész szám	[1, 2, …, 8]
rotary_table	karakterlánc	[“M”, “S”]
nest	egész szám	[1, 2, …, 8]
qib	egész szám	[0, 1]
error_code	egész szám	[100, 101, …, 899]
tray_row	egész szám	[0, 1, …, 10]
tray_column	egész szám	[0, 1, …, 10]

C.1.2. Közúti közlekedési bírságkezelési folyamat

A közúti közlekedési bírságkezelési folyamat egy olaszországi helyi rendőrséghez kapcsolódik. A folyamat DPN folyamatmodellje a C.2. ábrán látható, az átmenetekhez rendelt őrkifejezéseket pedig a C.3. táblázat tartalmazza. A folyamathoz publikusan elérhető eseménynaplóban [149] lévő eseményadatok a rendőrség által használt információs rendszerből származnak. A folyamatmodell és az eseménynapló részletes leírása a [13] publikációban és az [57] doktori értekezés 261–265. oldalain olvasható. A folyamat a következő tevékenységekből áll:

- *Create Fine*: A bírság kezdeti létrehozása az információs rendszerben.
- *Send Fine*: A bírságról szóló értesítés postai úton elküldésre kerül az elkövető számára.
- *Insert Fine Notification (Notification)*: Az értesítés megérkezik az elkövetőhöz.
- *Add Penalty*: További büntetés kerül felszámításra.
- *Payment*: Az elkövető által teljesített kifizetést nyilvántartásba veszik.
- *Send For Credit Collection*: A be nem fizetett bírságok esetében külön hitelbeszedési eljárás indul.
- *Insert Date Appeal to Prefecture (Appeal to Prefecture)*: Az elkövető fellebbez a bírság ellen a kapitányságon.
- *Send Appeal to Prefecture (Send Appeal)*: A felhívást a helyi rendőrség elküldi a kapitányságnak.

- *Receive Result Appeal from Prefecture (Receive Result)*: A helyi rendőrség megkapja a fellebbezés eredményét.
- *Notify Result Appeal to Offender (Notify Result)*: A helyi rendőrség tájékoztatja az elkövetőt a fellebbezés eredményéről.
- *Appeal to Judge*: Az elkövető fellebbez a bírság ellen a bíróságon.



C.2. ábra. A közúti közlekedési bírságkezelési folyamat DPN folyamatmodellje

C.3. táblázat. A közúti közlekedési bírságkezelési folyamat DPN folyamatmodelljében használt őrkifejezések

Átmenet	**Őrkifejezés**
Send Fine	$delaySend' < 90days$
Appeal to Judge	$delayJudge' < 60days$
Appeal to Prefecture	$delayPrefecture' < 60days$
Receive Result	$dismissal = NIL$
Send for Credit Collection	$payment < amount + expenses$
Inv1	$(dismissal \neq NIL) \vee (payment \geq amount \wedge points = 0)$
Inv2	$payment \geq amount + expenses$
Inv3	$payment \geq amount + expenses$
Inv4	$dismissal = G$
Inv5	$dismissal = NIL$
Inv6	$dismissal = \#$

A folyamat eseménynaplóiban használt attribútumok neve, rövidítése, értéktípusa, és leírása:

- *amount, am* (folytonos szám): A bírság fizetendő összege.
- *article, ar* (egész szám): Az olasz közúti közlekedési törvény cikkelyének száma, amelyet a szabálysértő megsértett.
- *dismissal, di* (karakterlánc): Jelző, amely jelzi, hogy a bírságot elutasították-e és hogyan. Több érték is lehetséges:
  - „#” azt jelzi, hogy a bírságot a rendőrkapitányság utasította el, és

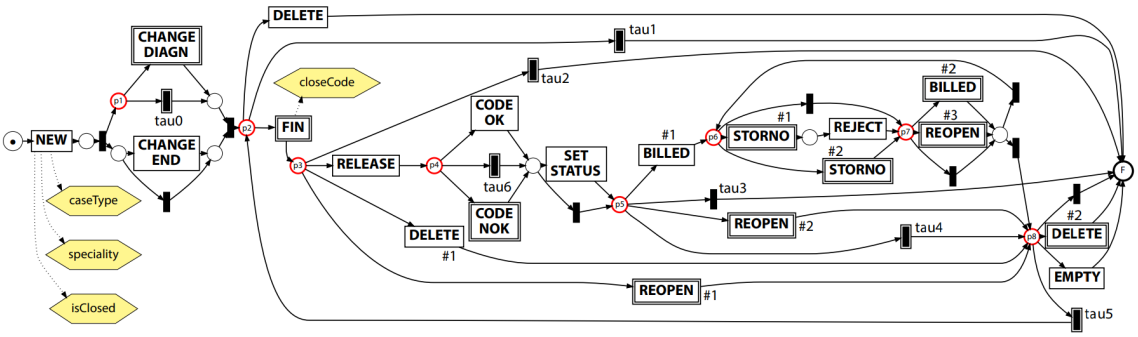
  - „NIL” azt jelzi, hogy a bírságot nem utasították el.
- *expense, ex* (folytonos szám): A postázási költségek miatt fizetendő kiegészítő összeg.
- *notificationType, nt* (karakterlánc): Egy kód, amely jelzi, hogy a bírság kire vonatkozik. Ha a tényleges elkövető ismeretlen (pl. nem állították meg), akkor a bírság a gépkocsi tulajdonosára vonatkozik. Az eseménynaplóban használt kódok a következők:
  - „P” (a gépkocsi tulajdonosa) vagy
  - „C” (a szabálysértést elkövető járművezető).
- *org:resource, or* (karakterlánc): Az ügyet kezelő alkalmazottat jelző kód.
- *paymentAmount, py* (folytonos szám): Az elkövető által egy tranzakció során kifizetett összeg.
- *points, po* (egész szám): Az elkövető jogosítványából levont büntetőpontok. Olaszországban minden járművezető 20 ponttal kezd a jogosítványán, és minden egyes kihágásért pontokat veszíthet. Azoknak a járművezetőknek, akik elveszítik az összes pontjukat, új vezetési vizsgát kell tenniük.
- *totalPaymentAmount (payment), pa* (folytonos szám): Az elkövető által fizetett teljes összeg.
- *vehicleClass, vc* (karakterlánc): Az elkövető által használt gépjármű típusa.

C.1.3. Kórházi számlázási folyamat

A kórházi számlázási folyamat egy hollandiai regionális kórházhoz kapcsolódik. A folyamat DPN folyamatmodellje a C.3. ábrán látható, az átmenetekhez rendelt őrkifejezéseket pedig a C.4. táblázat tartalmazza. A folyamathoz publikusan elérhető eseménynaplóban [150] lévő eseményadatok a kórház által használt információs rendszerből származnak. Az eseménynapló olyan eseményeket tartalmaz, amelyek a kórház által nyújtott orvosi szolgáltatások számlázásával kapcsolatosak. Az eseménynapló minden egyes nyomvonala azokat a tevékenységeket rögzíti, amelyek az egybegyűjtött orvosi szolgáltatások egy csomagjának számlázása érdekében kerültek végrehajtásra. Az eseménynapló nem tartalmaz információt a kórház által nyújtott tényleges orvosi szolgáltatásokról. A folyamatmodell és az eseménynapló részletes leírása az [57] doktori értekezés 325–330. oldalain olvasható.
A folyamat a következő tevékenységekből áll:

- *NEW*: Egy új számlázási csomag jön létre.
- *FIN*: A számlázási csomag lezárásra kerül, tehát már nem módosítható.
- *RELEASE*: A számlázási csomagot elküldik a biztosítótársaságnak.
- *CODE OK*: A bejelentési kódot sikeresen megkapták.
- *BILLED*: A számlázási csomagot kiszámlázták, tehát a számlát kiküldik.
- *CHANGE DIAGN*: Megváltozott a diagnózis, amelyen a számlázási csomag alapul.
- *DELETE*: A számlázási csomagot törlik.
- *REOPEN*: A számlázási csomagot újra megnyitják, így további orvosi szolgáltatásokat lehet hozzáadni, vagy meglévő szolgáltatásokat lehet törölni.
- *CODE NOK*: A bejelentési kódot hibaüzenettel kapják meg.
- *STORNO*: A számlázási csomagot visszavonják.
- *REJECT*: A biztosítónak küldött számlát elutasítják.

- *SET STATUS*: A státuszt (pl. új, lezárt stb.) manuálisan módosítják.
- *EMPTY*: A számlázási csomagot üressé nyilvánítják, mivel nem tartalmaz számlázható szolgáltatásokat.
- *MANUAL*: A számlázási csomagot kézzel módosítják egy nem szabványos rendszerből.
- *JOIN-PAT*: Két számlázási csomagot összekapcsolnak, mivel ugyanarra a betegre vonatkoznak.
- *CODE ERROR*: A bejelentési kódot nem sikerül megszerezni.
- *CHANGE END*: A számlázási csomag tervezett lejárati dátuma megváltozott.



C.3. ábra. A kórházi számlázási folyamat DPN folyamatmodellje

C.4. táblázat. A kórházi számlázási folyamat DPN folyamatmodelljében használt őrkifejezések

Átmenet	**Őrkifejezés**
CHANGE DIAGN	$caseType = B$
tau0	$caseType \neq B$
tau1	$speciality = K$
FIN	$speciality \neq K$
tau2	$closeCode = H$
REOPEN #1	$closeCode \neq H$
tau6	$caseType = F \vee (caseType \neq F \wedge caseType \neq C \wedge closeCode \neq A)$
CODE NOK	$(caseType \neq F \wedge caseType = C) \vee (caseType \neq F \wedge caseType \neq C \wedge closeCode = A)$
REOPEN #2	$caseType = B \vee (caseType \neq B \wedge closeCode \neq A)$
tau4	$caseType \neq B \wedge closeCode = A$
STORNO #1	$(isClosed = true \wedge closeCode \neq A) \vee isClosed \neq true$
STORNO #2	$isClosed = true \wedge closeCode = A$
BILLED	$isClosed = true$
REOPEN #3	$isClosed \neq true$
DELETE #2	$isClosed \neq true$
tau5	$isClosed = true$

A folyamat eseménynaplóiban használt attribútumok neve, értéktípusa, és leírása:

- *actOrange* (logikai érték): Olyan szolgáltatásokhoz kapcsolódóan használt jelző, amelyeket az általános egészségbiztosítás nem feltétlenül fedez.
- *actRed* (logikai érték): Olyan szolgáltatásokhoz kapcsolódóan használt jelző, amelyeket az általános egészségbiztosítás nem fedez.
- *blocked* (logikai érték): Egy olyan jelző, amelyet akkor használnak, ha a számlázás nem folytatódhat (azaz blokkolva van).
- *caseType* (karakterlánc): A számlázási csomag típusának kódja, amely befolyásolhatja annak kezelését.
- *closeCode* (karakterlánc): Egy számlázási csomag lezárásának több oka is lehet, ez az attribútum tárolja az alkalmazott kódot.
- *diagnosis* (karakterlánc): A számlázási csomagban használt diagnózis kódja.
- *flagA*, *flagB*, *flagC*, *flagD* (karakterlánc): Anonimizált jelzők.
- *isCancelled* (logikai érték): Jelző, amely azt jelzi, hogy a számlázási csomagot végül visszavonták-e.
- *isClosed* (logikai érték): Jelző, amely azt jelzi, hogy a számlázási csomagot végül lezárták-e.
- *msgCode* (karakterlánc): A „CODE NOK” tevékenység által visszaadott kód.
- *msgCount* (egész szám): A „CODE NOK” tevékenység által visszaadott üzenetek száma.
- *msgType* (karakterlánc): A „CODE NOK” tevékenység által visszaadott üzenetek típusa.
- *speciality* (karakterlánc): Az érintett orvosi szakterület kódja.
- *state* (karakterlánc): A számlázási csomag aktuális állapotát tárolja.
- *version* (karakterlánc): A használt szabályok verziójának kódja.

C.2. A javasolt MOCC módszer hatékonyságának vizsgálatának kimenetei

C.2.1. Az OCC vizsgálat eredményei

C.5. táblázat. Az OCC és az MOCC módszerek által adott online és offline igazítások teljes költsége az automatizált tekercs összeszerelési folyamat nem teljesen illeszkedő nyomvonalai esetén

Eset-azonosító	**Online igazítás**		**Offline igazítás**	
	OCC	**MOCC**	**OCC**	**MOCC**
287	0	0,2	0	0,2
386	0	0,2	0	0,2
429	0	0,2	0	0,2
2339	0	1	0	1
2726	0	1	0	1
3654	0	1	0	1
3685	0	1	0	1
3688	0	1	0	1
8306	2	2	4	5

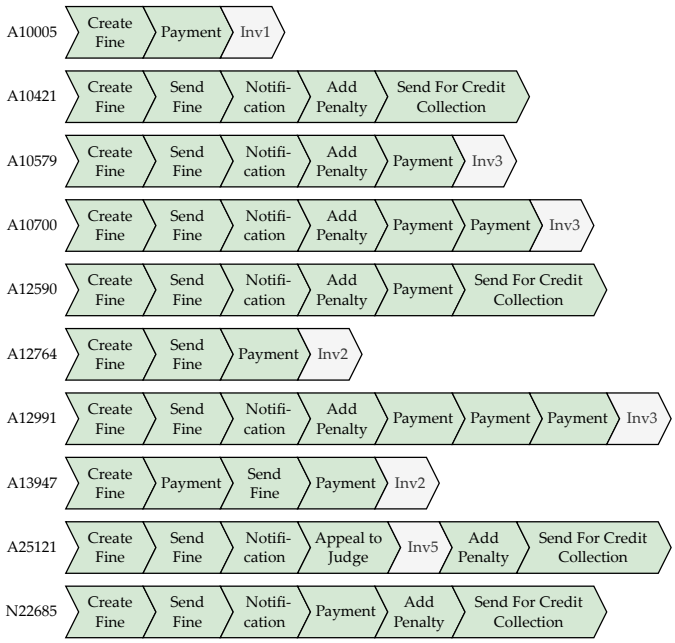
C.5. táblázat. Az OCC és az MOCC módszerek által adott online és offline igazítások teljes költsége az automatizált tekercs összeszerelési folyamat nem teljesen illeszkedő nyomvonalai esetén (folytatás)

Eset-azonosító	**Online igazítás**		**Offline igazítás**	
	OCC	**MOCC**	**OCC**	**MOCC**
8305	1	1	2	2,6
8307	2	2	4	5
8308	2	2	4	5
8309	2	2	4	5
8894	0	0,2	0	0,2
11121	0	0	4	5
13295	0	0,2	0	0,2
29642	0	0,1	6	7,5
29654	0	0	5	6,2
29655	0	0,1	6	7,5
36371	0	0,3	0	0,3
36370	0	0,3	0	0,3
36369	0	1,1	0	1,1
42013	0	1	0	1
42015	4	5,4	4	5,4
44891	0	0,2	0	0,2
44893	0	0,2	0	0,2
44894	0	0,2	0	0,2
44900	0	0,2	0	0,2
44913	0	0	0	1,6
44917	0	0,2	0	0,2
44974	0	0,2	0	0,2
44985	0	0,2	0	0,2
45007	0	0	0	1,6
45006	0	0	0	1,6
45021	0	0,2	0	0,2
45023	0	0,2	0	0,2
45043	0	1	0	1
45068	0	0,2	0	0,2
45180	0	1	0	1
45462	0	1	0	1
45734	0	1	0	1
46134	0	1	0	1
47494	0	0	3	3,8
47747	1	1	1	1
56746	0	1	0	1
57970	0	1	0	1
61281	0	0,2	0	0,2
68401	0	1	0	1
68415	0	0,2	0	0,2
71908	0	0,3	0	0,3
71914	0	0,3	0	0,3

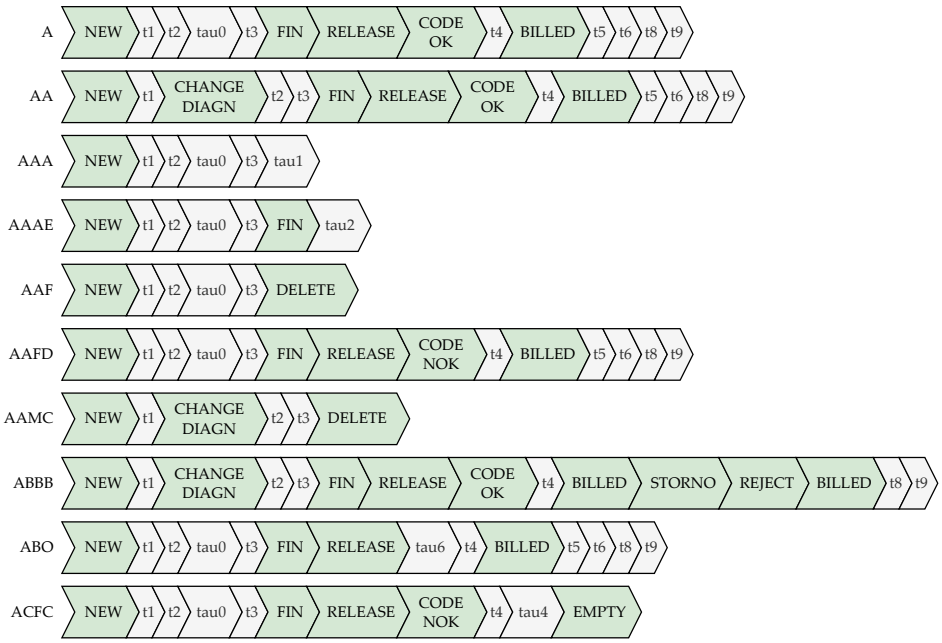
C.5. táblázat. Az OCC és az MOCC módszerek által adott online és offline igazítások teljes költsége az automatizált tekercs összeszerelési folyamat nem teljesen illeszkedő nyomvonalai esetén (folytatás)

Eset-azonosító	**Online igazítás**		**Offline igazítás**	
	OCC	**MOCC**	**OCC**	**MOCC**
71913	0	1,1	0	1,1
71910	0	1,1	0	1,1
71909	0	0,1	0	0,1
94599	0	0	3	3,8
94598	0	0	4	5
94597	0	0,1	5	6,3
94606	0	1	0	1
94608	0	1	0	1
94624	0	1	0	1
94632	0	1	0	1
94642	0	1	0	1
94673	0	0	2	2,6
94675	0	0,1	3	4
94674	0	0	4	5
94677	0	1,1	0	1,1
94735	0	0	1	1,4
94977	0	0,2	0	0,2
100306	0	1	0	1
114362	0	0	5	6,2
114361	0	0,1	6	7,5
128265	0	0,1	6	7,5
146534	0	1	0	1
146535	2	3,2	2	3,2
146538	2	3,2	2	3,2
167068	0	0,2	0	0,2
183454	0	0,2	0	0,2
184190	0	0,2	0	0,2
184197	0	0,2	0	0,2
187477	0	0,2	0	0,2
188242	0	1,1	0	1,1
188241	0	1,1	0	1,1
188240	0	1,1	0	1,1
196511	0	1	0	1
196519	0	1,1	0	1,1

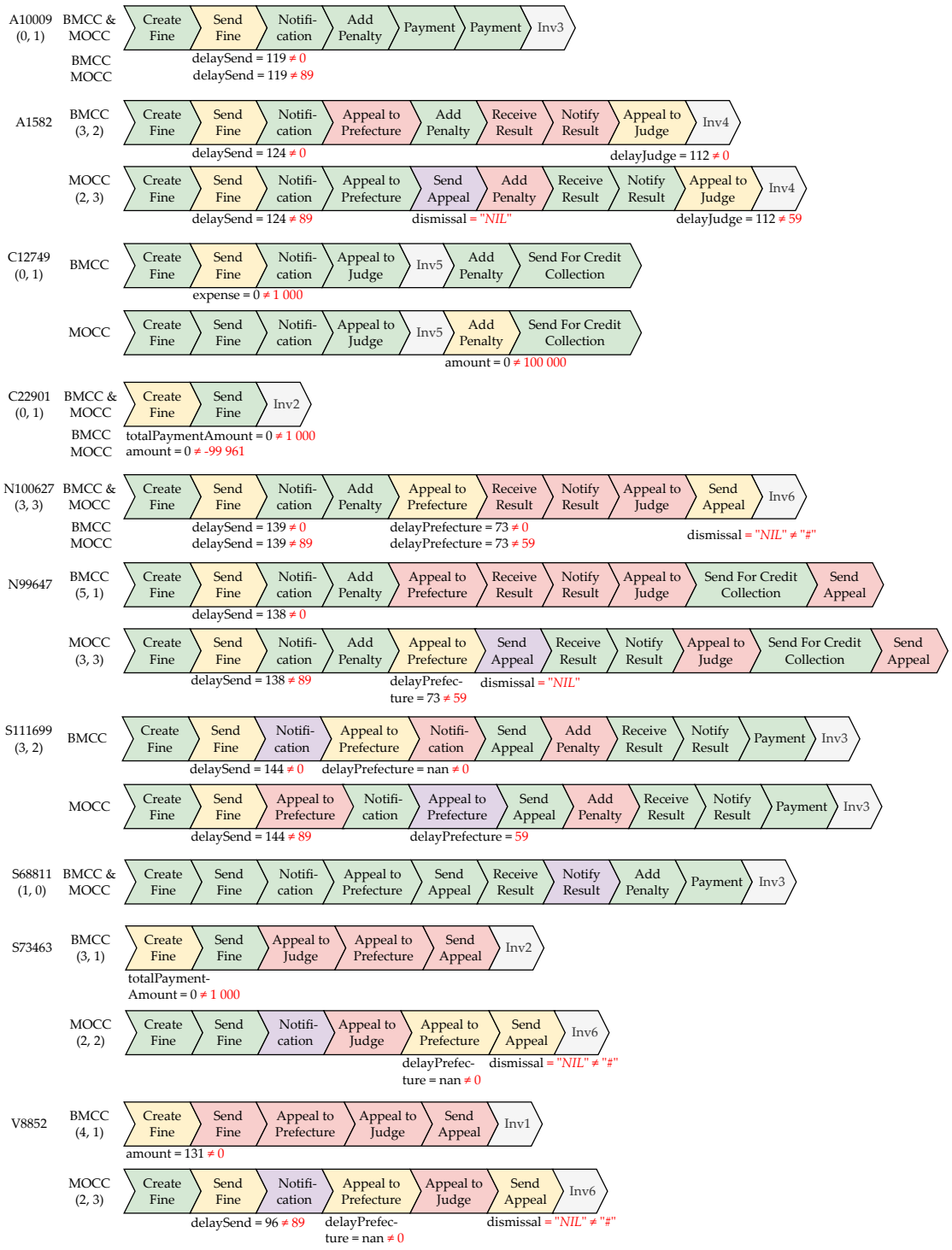
C.2.2. Az MCC vizsgálat eredményei



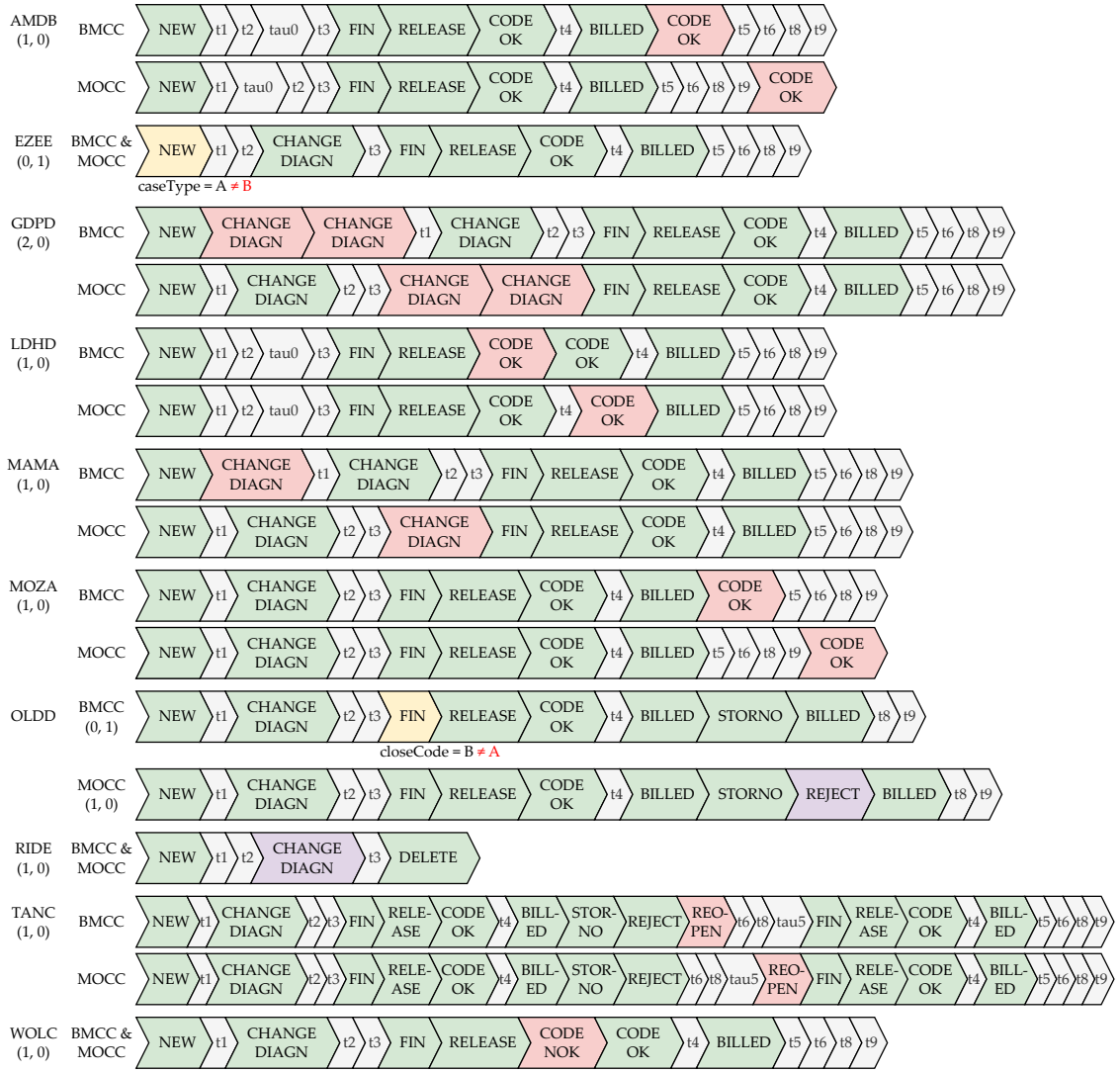
C.4. ábra. A BMCC és az MOCC módszerek által adott több perspektívás igazítások a közúti közlekedési bírságkezelési folyamat illeszkedő nyomvonalai esetén



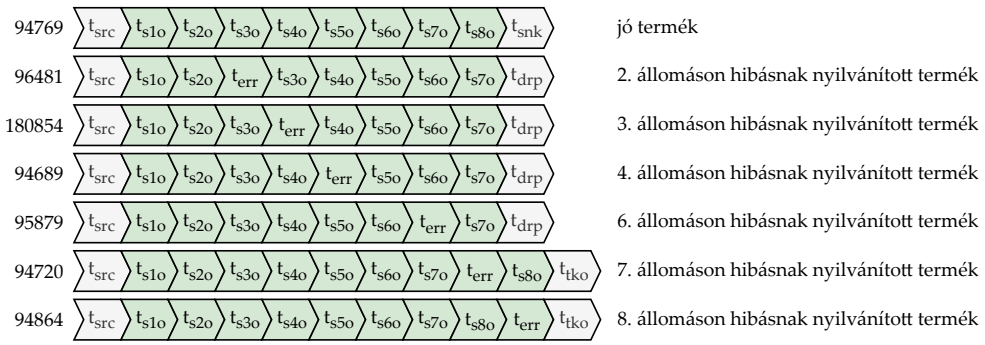
C.5. ábra. A BMCC és az MOCC módszerek által adott több perspektívás igazítások a kórházi számlázási folyamat illeszkedő nyomvonalai esetén



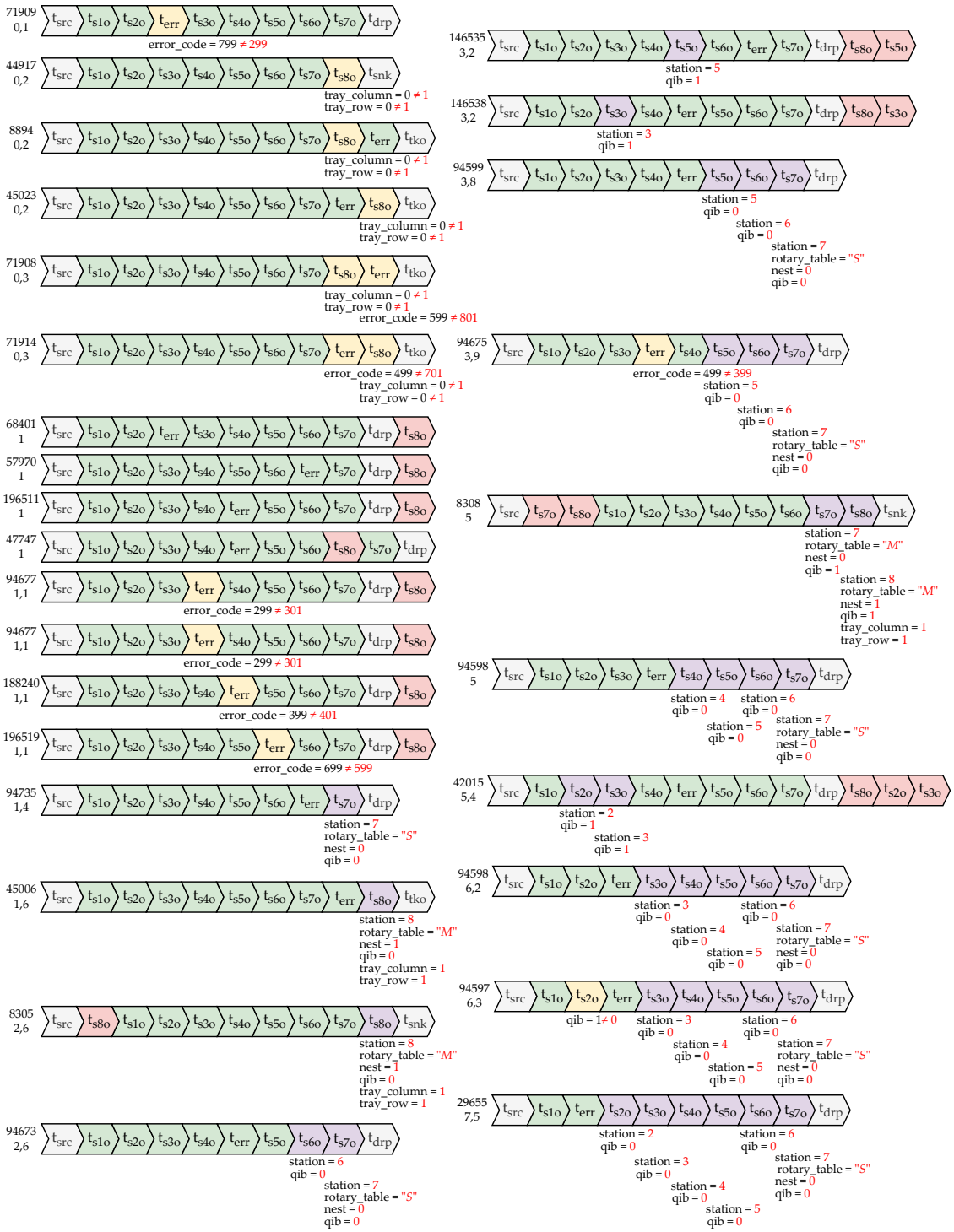
C.6. ábra. A BMCC és az MOCC módszerek által adott több perspektívás igazítások a közúti közlekedési bírságkezelési folyamat nem teljesen illeszkedő nyomvonalai esetén



C.7. ábra. A BMCC és az MOCC módszerek által adott több perspektívás igazítások a kórházi számlázási folyamat nem teljesen illeszkedő nyomvonalai esetén



C.8. ábra. A BMCC és az MOCC módszerek által adott több perspektívás igazítások az automatizált tekercs összeszerelési folyamat illeszkedő nyomvonalai esetén



C.9. ábra. Az MOCC módszer által adott több perspektívás igazítások az automatizált tekercs összeszerelési folyamat nem teljesen illeszkedő nyomvonalai esetén

C.2.3. Következtetések az automatizált tekercs összeszerelési folyamattal kapcsolatban

Az MOCC megoldás kimenete alapján a folyamatvégrehajtás során leggyakrabban előforduló hibás viselkedések azzal kapcsolatosak, hogy hogyan és mikor távolítják el a terméket a futószalagról. Egy késztermék esetében a „tray_row" és a „tray_column" értékeinek 0-nál nagyobbnak kell lenniük, ami azt jelzi, hogy a termék tálcára került elhelyezésre. Ellenkező esetben a kimeneti előtag-igazításban a 8. állomás műveleteire vonatkozó átmenet és esemény helytelen szinkronmozgás formájában jelenik meg, ami úgy értelmezhető, hogy a termék eldobásra került. Befejezetlen hibás termék esetében az utolsó rögzített eseménynek a 7. állomáson végzett műveleteknek kell lennie, ami azt jelenti, hogy a termék a 7. állomáson eldobásra került. Ha vannak rákövetkező események rögzítve, akkor azok eseménynapló mozgások formájában jelennek meg a kimeneti előtag-igazításban, ami úgy értelmezhető, hogy a termék nem a 7. állomáson került eldobásra. Ha azonban az utolsó rögzített esemény egy 7. állomást megelőző állomáshoz tartozik, akkor a teljes nyomvonal befejezetlen jó végrehajtásként jelenik meg a kimeneti előtag-igazításban, még akkor is, ha ez úgy értelmezendő, hogy a termék nem megfelelő állomáson került eldobásra.

Megjegyzendő, hogy a jó viselkedések helytelenül rossz viselkedésként is rögzítésre kerülhetnek. Emiatt, amikor a fent említett esetek a valóságban előfordulnak, a gépkezelőnek ellenőriznie kell, hogy a probléma forrása a terméket lecserélő eszköz, az információt szolgáltató érzékelő vagy az adatokat rögzítő szoftver. Az is lehetséges, hogy a probléma forrása egy nem modellezett esemény bekövetkezése (pl. a gépkezelő levett egy terméket a gyártósorról).

D. függelék

A „Statikus és dinamikus kapacitáskorlátos ív útvonaltervezési probléma” fejezethez tartozó függelékek

D.1. A lehetséges események elemzése

Ebben a függelékben összegyűjtöttem és elemeztem a lehetséges eseményeket egy DCARP példára és annak megoldására (azaz szolgáltatási tervére) gyakorolt hatásuk alapján. Minden egyes esemény kapcsán megvizsgáltam, hogy bekövetkezésük esetén szükség van-e az aktuális szolgáltatási terv módosítására és az újratervezésre, hogy megvalósítható tervet kapjunk. Ezt követően összehasonlítottam a D(C)ARP-ról szóló meglévő publikációkat aszerint, hogy milyen típusú dinamikus eseményeket vesznek figyelembe.

Feltételezések:

- Egy $a \in A$ ív $dc(a)$ áthaladási költségét a kapcsolódó útszakasz hossza (statikus érték) és az adott útszakasz forgalmi állapota (dinamikus érték) határozza meg.
- Egy $t \in T$ feladat ívének $sc(t)$ szolgáltatási költségét a $dc(t)$ áthaladási költség és a feladaton belüli részfeladatok költségei (dinamikus érték) határozzák meg. A feladaton belüli részfeladatok költségét az igény (azaz $dem(t)$) határozza meg. Ezért, ha az igény nő/csökken, a szolgáltatási költség is nő/csökken.
- A feladatokat tartalmazó útszakaszokat nem lehet lezárni (mivel az megoldhatatlanná tenné a példát).
- Csak olyan feladat nélküli útszakasz kerülhet lezárásra, aminek kiesésével nem változik meg a gráf elérhetőségi tulajdonsága (azaz egy v_i és v_j csúcs között mindig van út, ha $v_i, v_j \in V$).

A lehetséges váratlanul bekövetkező, példát módosító eseményeket két fő csoportba lehet osztani aszerint, hogy a probléma milyen típusú komponenséhez kapcsolódnak; az úthálózaton belül egy (vagy több) $a \in A$ útszakaszhoz (azaz egy vagy

több ívhez) vagy egy $h \in H$ járműhöz. Az úthálózathoz kapcsolódó lehetséges eseményeket a D.1. táblázat, a járművekhez kapcsolódó lehetséges eseményeket pedig a D.2. táblázat foglalja össze. Az „M." oszlopban az van jelölve, hogy szükséges-e a jelenlegi szolgáltatási terv módosítása, az „Út." oszlopban pedig az van jelölve, hogy szükség van-e újratervezésre. A csillag feltételes választ jelent; azaz a módosítás/újratervezés csak akkor szükséges, ha az eseménynek a problémára és szolgáltatási tervre gyakorolt hatása megfelel egy feltételnek. Látható, hogy néhány esetben (pl. az „útlezárás" és a „feladat lemondás" események esetén) az aktuális szolgáltatási terv frissítése egyszerű algoritmusokkal könnyen megvalósítható, így újratervezésre nincs szükség. Néhány másik esetben (pl. a „feladat megjelenés", az „igény növekedés" és a „jármű meghibásodás" események esetén) az aktuális szolgáltatási terv frissítése összetettebb algoritmust igényel, így újratervezésre van szükség. A továbbiakban ezeket az eseményeket együttesen **kritikus események**nek nevezem.

D.1. táblázat. Az úthálózathoz kapcsolódó lehetséges események és azok problémára gyakorolt hatása

Esemény	**A problémára gyakorolt hatás**	**M.**	**Út.**
Útlezárás	Az érintett ív(ek) *dc* áthaladási költsége ($x \in \mathbb{R}_{\geq 0}$ értékről) ∞ értékre nő.	✓⋆	
Útmegnyitás	Az érintett ív(ek) *dc* áthaladási költsége (∞ értékről) $x \in \mathbb{R}_{\geq 0}$ értékre csökken.		
Forgalom csökkenés	Az érintett ív(ek) *dc* áthaladási költsége csökken. A még nem elvégzett feladatok esetében az *sc* szolgáltatási költség is csökken.		
Forgalom növekedés	Az érintett ív(ek) *dc* áthaladási költsége nő. A még nem elvégzett feladatok esetében az *sc* szolgáltatási költség is nő.		
Feladat lemondás	Az érintett ív(ek) a T feladathalmazból eltávolításra kerül(nek).	✓	
Feladat megjelenés	Az érintett ív(ek) a T feladathalmazhoz hozzáadásra kerül(nek).	✓	✓
Igény csökkenés	Az érintett ívfeladat(ok) *dem* igénye és *sc* szolgáltatási költsége csökken.		
Igény növekedés	Az érintett ívfeladat(ok) *dem* igénye és *sc* szolgáltatási költsége nő.	✓⋆	✓⋆

D.1.1. Az úthálózathoz kapcsolódó lehetséges események elemzése

Ha egy utat lezárnak, azaz bekövetkezik egy „**útlezárás**" esemény, akkor az érintett útszakasz(ok) többé nem lesz(nek) átjárható(ak). Mivel ez a helyzet csak átmeneti, a kapcsolódó ív(ek) eltávolítása helyett csak az áthaladási költség(ek) módosul(nak) végtelen (∞) értékre. Ettől még nem lesz szükség újratervezésre, de az aktuális szolgáltatási terv megvalósíthatatlanná válhat, ha az érintett ív(ek)en a terv szerint át kell haladni. Ilyen esetben újra kell számítani az érintett feladatok közötti legrövidebb utat az új költségértékekkel, majd a kérdéses útvonalterveket frissíteni kell.

D.2. táblázat. A járművekhez kapcsolódó lehetséges események és azok problémára gyakorolt hatása

Esemény	**A problémára gyakorolt hatás**	**M.**	**Út.**
Jármű meghibásodás	A rendelkezésre álló járművek száma eggyel csökken. Ha az érintett járműnek van(nak) végrehajtatlan feladat(ai), az(oka)t egy másik járműnek kell kiszolgálnia.	✓⋆	✓⋆
Jármű beüzemelés	A rendelkezésre álló (szabad) járművek száma eggyel nő.		

Amennyiben a két feladat közötti új legrövidebb útvonal az útvonalterv egy vagy több még végre nem hajtott feladatát tartalmazza, a kérdéses feladat(ok) beilleszthető(ek) oda. Ekkor a feladatok végrehajtási sorrendje megváltozik, ezért az érintett feladatok közötti legrövidebb útvonalakat újra kell számítani. Előfordulhat, hogy ezektől a módosításoktól még nem lesz a megoldás optimális, de megvalósítható lesz. Újratervezéssel lehet, hogy tovább lehetséges csökkenteni a teljes költséget.

Ha egy utat megnyitnak, azaz bekövetkezik egy „**útmegnyitás**” esemény, akkor az érintett útszakasz(ok) újra átjárható(ak) lesz(nek). Ekkor az áthaladási költség(ek) az útszakasz(ok) hossza és forgalmi állapota alapján új érték(ek)re módosul(nak). Az „útmegnyitás” eseményhez hasonlóan ekkor sincs szükség a jelenlegi szolgáltatási terv módosítására és újratervezésre sem. Ugyanakkor az újratervezés javíthatja a jelenlegi szolgáltatási tervet, ha egy vagy több útvonaltervben a megnyitott útszakasz(ok) használata csökkenti a teljes költséget.

Ha egy úton vagy útszakaszon csökken a forgalom, azaz bekövetkezik egy „**forgalom csökkenés**” esemény, akkor az csökkenti a kapcsolódó ív(ek) áthaladási költségét. Továbbá, ha az ív(ek)en van még nem elvégzett feladat, akkor az ív(ek) kiszolgálási költségét is csökkenti. Ekkor nincs szükség a jelenlegi szolgáltatási terv módosítására és újratervezésre sem. Az újratervezés azonban javíthatja a jelenlegi szolgáltatási tervet, ha egy vagy több útvonaltervben az érintett útszakasz(ok) használata csökkenti a teljes költséget.

Ha egy úton vagy útszakaszon nő a forgalom, azaz bekövetkezik egy „**forgalom növekedés**” esemény, akkor az növeli a kapcsolódó ív(ek) áthaladási költségét. Továbbá, ha az ív(ek)en van még nem elvégzett feladat, akkor az ív(ek) kiszolgálási költségét is növeli. Ekkor nincs szükség a jelenlegi szolgáltatási terv módosítására és újratervezésre sem. Azonban az újratervezés javíthatja a jelenlegi szolgáltatási tervet, ha egy vagy több útvonaltervben az érintett útszakasz(ok) helyett más útszakasz(ok) használata csökkenti a teljes költséget. Ha a teljes költség túlzott mértékben megnövekedett több szolgáltatási terv által érintett útszakaszon is, azaz ha egy (vagy több) ilyen esemény hatása az „útlezárás” eseményhez hasonló hatást fejt ki nagyobb területen, akkor javasolt az újratervezés és így a szolgáltatási terv módosítása is.

Ha egy feladat visszavonásra kerül, azaz bekövetkezik egy „**feladat lemondás**” esemény, akkor az érintett ív (és ha van, akkor annak inverze is) a feladathalmazból eltávolításra kerül. Ettől még nem lesz szükség újratervezésre, de a jelenlegi szolgáltatási terv elavulttá válik. A kérdéses feladatot el kell távolítani a szolgáltatási terv érintett útvonaltervéből, majd a szolgáltatási tervben frissíteni kell az eltávolított

feladatot megelőző és követő feladat közötti útvonalat a kettő közötti legrövidebb útvonallal. Az újratervezés tovább javíthatja a frissített szolgáltatási tervet, ha a feladat eltávolítása költséghatékonyabb terveket tett megvalósíthatóvá.

Ha egy új feladat jelenik meg egy olyan útszakaszon, amelyen korábban nem volt még feladat (vagy volt, de az(ok) már végrehajtásra került(ek)), azaz bekövetkezik egy „**feladat megjelenés**" esemény, akkor a kapcsolódó ív (és ha van, akkor annak inverze is) hozzáadódik a feladathalmazhoz, valamint a megadott szolgáltatási költség és igény hozzárendelődik ehhez az ívhez. Ekkor a jelenlegi szolgáltatási terv elavulttá válik, ezért az új feladatot a szolgáltatási terv már egy létező vagy új útvonaltervéhez hozzá kell adni. Mivel nem egyértelmű, hogy pontosan melyik útvonaltervbe és azon belül melyik pozícióba kell beszúrni a feladatot, ezért újratervezés szükséges.

Ha egy útszakaszon csökken az igény, azaz bekövetkezik egy „**igény csökkenés**" esemény, akkor a kapcsolódó ívfeladathoz (és ha van, akkor annak inverzéhez is) tartozó igény is csökken, így a hozzátartozó szolgáltatási költség is csökkenhet. Ez nincs ráhatással a szolgáltatási terv megvalósíthatóságára és emiatt újratervezésre sincs szükség. Ugyanakkor az újratervezés javíthatja a jelenlegi szolgáltatási tervet, ha az érintett feladatot egy másik jármű is ki tudja szolgálni, de alacsonyabb teljes költséggel.

Ha egy útszakaszon nő az igény, azaz bekövetkezik egy „**igény növekedés**" esemény, akkor a kapcsolódó ívfeladathoz (és ha van, akkor annak inverzéhez is) tartozó igény is nő, valamint a hozzátartozó szolgáltatási költség is növekedhet. Ez akkor teszi a jelenlegi megoldást megvalósíthatatlanná, illetve akkor teszi szükségessé az újratervezést, ha a feladat igénye olyan mértékben megnő, hogy az útvonaltervben szereplő feladatok összes igénye meghaladja a jármű kapacitását.

D.1.2. A járművekhez kapcsolódó lehetséges események elemzése

Ha egy jármű működése leáll, azaz bekövetkezik egy „**jármű meghibásodás**" esemény, akkor a szolgálatban lévő járművek száma eggyel csökken. Ez a jelenlegi szolgáltatási tervet megvalósíthatatlanná teszi és újratervezést tesz szükségessé, de csak akkor, ha az érintett jármű útvonaltervében vannak még nem elvégzett feladatok.

Ha egy korábban meghibásodott járművet sikeresen megjavítanak és szolgálatba visszahelyeznek, azaz bekövetkezik egy „**jármű beüzemelés**" esemény, akkor a szolgálatban lévő járművek száma eggyel nő. Ez nincs ráhatással a szolgáltatási terv megvalósíthatóságára és emiatt újratervezésre sincs szükség. Azonban az újratervezés javíthatja a jelenlegi szolgáltatási tervet, ha a beüzemelt jármű a többi járműnél alacsonyabb teljes költséggel tudja végrehajtani a még végrehajtásra váró feladatok egy részét.

D.1.3. A meglévő D(C)ARP megközelítések összehasonlítása

A D.3. táblázatban a D(C)ARP-hoz kapcsolódó szakirodalom és a problémát módosító lehetséges események szerepelnek. A táblázat tartalma nagyrészt egy DARP-okról szóló irodalmi áttekintésen alapul ([32]). A fejlécben szereplő számok a publikációkra való hivatkozások. Az X-ek azt jelzik, hogy a hivatkozott publikáció az adott

eseményt figyelembe veszi-e. Az esemény nevének jobb oldalán lévő csillag azt jelzi, hogy az eseményt a hivatkozott publikációkban esetleg másképp nevezik, de ennek ellenére azonos hatást gyakorol a problémára. A kritikus események félkövér betűkkel vannak jelölve. Szürke színnel azok az események vannak jelölve, amiket egyik publikáció sem vesz figyelembe. Látható, hogy csak a DCARP-ról szóló legfrissebb publikáció [46] veszi figyelembe az összes kritikus eseményt. Emellett az az a publikáció, amely a legtöbb lehetséges eseményt figyelembe veszi.

D.3. táblázat. A D(C)ARP-hoz kapcsolódó szakirodalom és a problémát módosító lehetséges események

Esemény	**[135]**	**[138]**	**[136]**	**[139]**	**[140]**	**[141]**	**[134]**	**[137]**	**[46]**
Útlezárás		X	X					X	X
Útmegnyitás									X
Forgalom csökkenés*									X
Forgalom növekedés*		X					X		X
Feladat lemondás	X	X		X					
Feladat megjelenés	**X**	**X**			**X**	**X**			**X**
Igény csökkenés									
Igény növekedés		**X**	**X**						**X**
Jármű meghibásodás								**X**	**X**
Jármű beüzemelés									

D.2. Az alap ABC algoritmus

Inicializálási fázis

Az inicializálási fázisban a paraméterek és a populáció inicializálása történik. Az ABC algoritmus paraméterei a következőképpen határozhatók meg:

- *sn*: a táplálékforrások száma, amely egyben a dolgozó méhek és a megfigyelő méhek száma is (azaz minden táplálékforrásra csak egy dolgozó méh jut);
- *limit*: a kísérletek száma, amely után egy táplálékforrást feltételezhetően elhagynak;
- egy befejezési kritérium.

A populáció inicializálása úgy történik, hogy véletlenszerűen *sn* számú táplálékforrást generálunk, és mindegyikhez egy-egy dolgozó méhet rendelünk. A dolgozó méhek felmérik e megoldások megfelelőségét.

Dolgozó méh fázis

Ebben a fázisban minden egyes dolgozó méh a hozzá tartozó x_i táplálékforrás szomszédságában egy x_j új táplálékforrást generál. Miután x_j megvan, kiértékelésre, majd összehasonlításra kerül x_i-vel. Ha az x_j nektármennyisége megegyezik vagy nagyobb, mint az x_i-é, akkor x_j lecseréli x_i-t és a populáció új tagjává válik, ellenkező esetben x_i megmarad. Más szóval, egy mohó kiválasztási módszert alkalmazunk a régi és az új lehetséges megoldások között.

Megfigyelő méh fázis

Egy megfigyelő méh kiértékeli az összes dolgozó méhtől kapott nektárinformációt, majd kiválaszt egy x_i táplálékforrást a következő kifejezéssel kiszámított p_i valószínűségi érték függvényében:

$$p_i = \frac{fit_i}{\sum_{j=1}^{sn} fit_j} \tag{D.1}$$

ahol fit_i az i-edik táplálékforrás (azaz x_i) nektármennyisége. Minél nagyobb a fit_i értéke, annál nagyobb a valószínűsége annak, hogy az i-edik táplálékforrás kerül kiválasztásra.

Miután a megfigyelő kiválasztotta az x_i táplálékforrást, egy lokális keresési művelet segítségével módosítja az x_i értékét. A lokális keresési művelet véletlenszerűen kiválaszt egy x_j helyet az x_i szomszédságában. A dolgozó méhekhez hasonlóan, ha az x_j jobb vagy azonos nektármennyiséggel rendelkezik, mint x_i, akkor x_j leváltja x_i-t a populációban.

Felderítő méh fázis

Ha egy x_i táplálékforrást nem lehet tovább javítani egy előre meghatározott számú próbálkozáson belül, akkor a táplálékforrást elhagyottnak tekintjük, és a kapcsolódó dolgozó méh felderítővé válik. A felderítő véletlenszerűen hoz létre egy táplálékforrást.

Az alap ABC algoritmusban minden ciklusban legfeljebb egy felderítő méh megy ki, hogy új táplálékforrást keressen.

D.3. Mozgási műveletek a CARP-hoz

Inverziós művelet

Az inverziós művelet véletlenszerűen kiválaszt egy $t \in T$ feladatot a bemeneti megoldásból. Ha ennek a feladatnak van inverze (azaz $\exists inv(t) \in T$), akkor a művelet t-t $inv(t)$-re cseréli a megoldáson belül. Ellenkező esetben a bemeneti megoldást adja vissza.

Betoldási művelet

A betoldási művelet véletlenszerűen kiválaszt egy $t_1 \in T$ feladatot a feladathalmazból, majd a bemeneti megoldáson belül egy másik véletlenszerűen kiválasztott $t_2 \in T$ feladat elé vagy után helyezi (azaz betoldja) azt. A kiválasztott feladatok lehetnek azonos vagy különböző útvonaltervből is, de maguk a feladatok nem lehetnek azonosak (beleértve az inverzüket is). Azaz $t_1 \neq t_2$, és ha $\exists inv(t_2) \in T$, akkor $t_1 \neq inv(t_2)$.

Csere művelet

A csere művelet véletlenszerűen kiválaszt két feladatot (t_1 és t_2, ahol $t_1, t_2 \in T$) a bemeneti megoldásban, majd kicseréli őket egymással. A betoldási művelethez hasonlóan a kiválasztott feladatok lehetnek azonos vagy különböző útvonaltervből is, de maguk a feladatok nem lehetnek azonosak.

A művelet olyan lehetséges kimeneti megoldásokat is létrehoz, amelyek a kiválasztott feladat(ok) helyett azok inverzét tartalmazza (amennyiben az létezik). Az összes lehetséges (maximum négy) kombinációt figyelembe veszi, amik közül azt választja kimeneti megoldásként, amelynek a legkisebb az összköltsége.

Kétopciós művelet

A kétopciós művelet először véletlenszerűen kiválaszt két útvonaltervet (pl. r_1 és r_2) a megoldásból. Ha a kiválasztott két útvonalterv megegyezik (azaz $r_1 = r_2$), akkor az útvonaltervben egy részútvonaltervet (azaz az útvonalterv egy részét) véletlenszerűen kiválaszt, és annak irányát megfordítja. Ha a kiválasztott két útvonalterv különbözik (azaz $r_1 \neq r_2$), akkor ezt a két útvonaltervet véletlenszerűen ketté vágja, négy részútvonaltervet hozva létre. A négy részútvonalterv újbóli összeillesztésével két új lehetséges kimeneti megoldást hoz létre, amik közül a legkisebb összköltséggel rendelkezőt választja ki kimeneti megoldásként. Például, az r_1 és r_2 útvonalterveket r_{11} r_{12} és r_{21} r_{22} részútvonaltervekre vágja. Ezekből két új megoldást kapunk a következő módon történő összeillesztésükkel:

1. r_{11} r_{22} és r_{21} r_{12}, valamint
2. r_{11} fordított r_{21} és fordított r_{12} r_{22}.

Egyesítés-szétválasztás művelet

Amint azt már említettem, az egyesítés-szétválasztás művelet nagyobb változtatásokat tud végrehajtani a megoldásban (pl. megváltoztathatja az összes feladat sorrendjét egy vagy több útvonalterven belül), ezért nagy lépésméretű műveletnek számít. Ez a művelet véletlenszerűen kiválaszt x számú különböző útvonaltervet a bemeneti megoldásban, ahol x egy véletlen pozitív egész szám ($1 \leq x \leq |S|$). A kiválasztott útvonaltervek feladatainak egy listába történő egyesítésével a feladatok rendezetlen listáját kapja, majd ezt a rendezetlen listát egy útvonal-pásztázó heurisztikával rendezi (pl. [118], amelyet ebben a munkában is használok). A kapott rendezett listát ezután optimálisan új útvonaltervekre válassza szét Ulusoy szétválasztási eljárása [120] segítségével.

A rendezett listát az útvonal-pásztázó heurisztika a következőképpen építi fel. Először egy üres útvonal tervet inicializál, majd az érintett feladatokat egyenként hozzáadja az aktuális útvonalhoz, amíg a rendezetlen listában nem marad feladat. Minden egyes iterációban csak azokat a feladatokat veszi figyelembe, amelyek a kapacitási korlátozás megsértése nélkül adhatók hozzá. Ha nincs ilyen feladat, akkor a telephelyet adja hozzá, és egy új útvonalat inicializál (ez lesz az új aktuális útvonal). Amikor egy feladat vagy a telephely hozzáadásra kerül az aktuális útvonal végéhez, akkor ez a köztük lévő legrövidebb útvonallal történik. Ha több feladat is hozzáadható, akkor az aktuális útvonal végéhez legközelebb eső feladat kerül hozzáadásra. Ha több olyan feladat van, amire ez igaz, akkor a következő szabályok valamelyikét kell alkalmazni a következőként hozzáadandó feladat kiválasztásához:

1. a $head(t)$ és v_0 közötti távolság maximalizálása;
2. a $head(t)$ és v_0 közötti távolság minimalizálása;
3. a $\frac{dem(t)}{sc(t)}$ kifejezés maximalizálása;

4. a $\frac{dem(t)}{sc(t)}$ kifejezés minimalizálása;

5. ha a jármű kevesebb, mint félig van tele, akkor az 1. szabály alkalmazása szükséges, egyébként a 2. szabályt alkalmazzuk.

A fenti szabályokban t ($t \in T$) egy feladat, v_0 ($v_0 \in V$) pedig a telephely. Egy lefutásban egyszerre csak az egyik szabály használható, ezért az útkeresési heurisztikát ötször kell lefuttatni. Ez öt rendezett listát eredményez.

Ulusoy szétválasztási eljárása az öt rendezett lista útvonaltervekre történő felosztásával öt új lehetséges kimeneti megoldást hoz létre. Az eljárás működését legjobban a [151] foglalja össze. Az eljárás a rendezett listából az irányított aciklikus gráf (*Directed Acyclic Graph*, röviden *DAG*) létrehozásával kezdődik. A DAG egy olyan gráf, amelynek ívei egy óriási túra megvalósítható résztúráit jelölik. Ezután kiszámítja a legrövidebb utat a gráfon belül, ami megadja az óriási túra optimális felosztását megvalósító útvonaltervet. Utolsó lépésként a bemeneti megoldás érintetlen útvonalterveiből és az eljárás által visszaadott útvonaltervekből új megoldásjelöltet hoz létre. Az egyesítés-szétválasztás művelet az öt lehetséges megoldás közül azt választja kimeneti megoldásként, amelynek a legkisebb az összköltsége.

D.4. A felfedezésre összpontosító CARP-ABC algoritmus számítási komplexitásának elemzése

Ebben a szakaszban a felfedezésre összpontosító CARP-ABC algoritmus számítási komplexitása kerül kifejtésre. A számítási komplexitást a nagy O jelöléssel fejezzük ki. Az egyszerűség kedvéért közelítéseket használok, és a konstans értékeket elhagyom. A teljes algoritmus számítási komplexitása a megadott paraméterértékektől és a bemeneti CARP példa komplexitásától függ, elsősorban n-től (azaz a végrehajtandó feladatok számától).

Inicializálási fázis

Az inicializálási fázis számítási komplexitása $O(n_{cs} * n + n_{cs})$, ahol az $O(n)$ az RSG algoritmus komplexitása, az $O(n_{cs})$ pedig a legjobb megoldás kiválasztásának komplexitása. Az $O(n)$ szorozva van n_{cs} értékével, mivel az RSG-t n_{cs} alkalommal kell végrehajtani a kezdeti populáció létrehozásához.

Az RSG algoritmuson belül a feladatazonosítók véletlenszerű permutációjának létrehozásának bonyolultsága $O(n)$, feltételezve, hogy a Fisher-Yates keverési algoritmust [152] használjuk hozzá. A permutáció generálása után az algoritmus minden egyes elemen végig megy, aminek szintén $O(n)$ a bonyolultsága. Az RSG algoritmus komplexitása tehát $O(2 * n)$ körül van, ami a konstans szorzó elhagyásával $O(n)$.

Dolgozó méh fázis

A dolgozó méh fázis számítási komplexitása $O(n_{cs} * n_{lsl} * (n + \log n + n_{\max}) + n_{cs})$, amelyben a lokális keresés komplexitása $O(n_{lsl}) * (\log n + n + n_{\max})$. A valószínűségszámítás bonyolultsága $O(n_{cs})$ (feltételezve, hogy a megfelelőségi értékek összegét az algoritmus csak egyszer számolja ki). Az $O(n_{lsl} * (\log n + n + n_{\max}))$ megszorozódik n_{cs} értékével, mivel a lokális keresést a populáció összes n_{cs} tagjára végre kell hajtani.

A lokális keresésen belül az inverziós művelet bonyolultsága $O(n)$, a részútvonalterv művelet bonyolultsága pedig $O(\log n + n + n_{\max})$. A részútvonalterv műveleten belül az útvonalterv kiválasztásának bonyolultsága $O(\log n)$, mivel a legrosszabb esetben $|S| = n$ (azaz minden feladat külön útvonalon van). Az útvonalterv kiválasztása után a művelet egyik módszere kerül végrehajtásra. A módszerek közül a részútvonalterv forgatási módszer a legbonyolultabb, aminek a komplexitása $O(n + n_{\max})$, mivel a legrosszabb esetben $l_k = n$ (azaz csak egy útvonalterv van a megoldásban).

Megfigyelő méh fázis

A megfigyelő méh fázis számítási komplexitása $O(n_{cs} + n_{cs} * (\log n + n + n_{lsl} * (n + \log n + n_{\max})))$, amelyben a megoldás kolóniából való kiválasztásának bonyolultsága $O(k * n_{cs})$ vagy $O(n_{cs})$, ha a k konstans elhagyható. A fő keresés bonyolultsága $O(n_{cs} * (\log n + n + n_{lsl} * (n + \log n + n_{\max})))$.

A fő keresésen belül az egyesítés-szétválasztás művelet komplexitása $O(\log n + n)$, mivel az útvonaltervek számának kiválasztásának a komplexitása $O(\log n)$, a művelet többi komponensének (azaz az útvonaltervek kiválasztása, az érintett feladatok összegyűjtése és az RPSH végrehajtása) a komplexitása pedig $O(n)$. Az RPSH bonyolultsága $O(n)$, mivel a legrosszabb esetben az összes n feladat érintett a megoldásban. Miután az egyesítés-szétválasztás művelet visszaad egy megoldást, a keresés e megoldás körül történik. Ennek a keresésnek a komplexitása $O(n_{lsl} * (n + \log n + n_{\max}))$, mivel a részútvonalterv művelet bonyolultsága $O(\log n + n + n + n_{\max})$, a többi művelet (azaz az inverziós, a betoldási, a csere és a kétopciós műveletek) bonyolultsága pedig $O(n)$.

Felderítő méh fázis

A felderítő méh fázis számítási komplexitása $O(n_{cs} * n + n_{cs})$, mivel a legrosszabb esetben az összes megoldást le kell cserélni a kolóniában (n_{sal} érték átlépése esetén), így az RSG n_{cs} alkalommal kerül végrehajtásra. Ezután a kolóniából kiválasztásra kerül a legjobb megoldás, amelynek a komplexitása $O(n_{cs})$.

Teljes algoritmus

A teljes CARP-ABC algoritmus számítási komplexitása az inicializálási fázis komplexitásából és a többi fázis n_{mi} értékkel való megszorzásából tevődik össze, mivel a legrosszabb esetben az algoritmus az iterációk maximális számának eléréséig fut. Ha a duplikációkat eltávolítjuk, akkor ez a következő: $O(n_{cs} * n + n_{cs} + n_{mi} * (n_{cs} * n_{lsl} * (n + \log n + n_{\max}) + n_{cs} + n_{cs} * (\log n + n + n_{lsl} * (n + \log n + n_{\max}))))$.

Ha a CARP-ABC algoritmus és a részútvonalterv művelet paraméterei rögzített értékre vannak beállítva, akkor a számítási komplexitás a következő: $O(n + \log n)$. A CARP-ABC algoritmus időkomplexitása tehát többnyire lineáris, de tartalmaz logaritmikus időkomplexitású komponenseket (pl. egy útvonalterv kiválasztása).

D.5. Az esemény generátort használó adat-vezérelt DCARP keretrendszer működése

Ebben a szakaszban az esemény generátort használó adat-vezérelt DCARP keretrendszer fő algoritmusa, az esemény létrehozó modul, az eseménykezelő modul és az eseménykezelő algoritmusai, valamint az újratervező modul részletes működése kerül bemutatásra.

D.5.1. A fő algoritmus

Ebben a szakaszban a fő algoritmus (17. algoritmus) kerül bemutatásra.

Algoritmus 17: *dataDrivenDCARPFramework* (Adat-vezérlet DCARP keretrendszer)

```
   input  : I_0 = (V, v_0, A, T, n, w, q, H, head, tail, dc, inv, dem, sc, mdc)
1  begin
2      I, S ← initializationModule(I_0);   // probléma inicializálása, kezdeti szolgáltatási terv létrehozása
3      while true do
4          e ← eventGeneratorModule(I, S);                          // esemény generálása
5          I, S, S̄, T_k, reroute ← eventHandlerModule(I, S, e);     // esemény kezelése
6          if reroute = true then
7              I, S ← replanningModule(I, S, S̄, T_k);               // szolgáltatási terv újratervezése
8          if |H_f| + |H_e| = |H| ∧ n = 0 then                      // szimuláció befejezése
9              return;
```

Az algoritmus bemenetként egy kezdeti I_0 DCARP példát vár, ami lényegében egy statikus CARP példa. Első lépésként az **inicializációs modul** az I_0 példa alapján inicializálja a problémát és létrehozza a kezdeti szolgáltatási tervet (2. sor). Ekkor kerülnek a kezdeti példából hiányzó DCARP komponensek is inicializálásra és hozzáadásra a példához.

Az inicializálást követően megkezdődik a szimuláció, azaz az események generálása az **esemény létrehozó modul** segítségével (4. sor). Az események egyesével kerülnek létrehozásra és feldolgozásra, mindig az aktuális I példa és annak S megoldása alapján. Az esemény feldolgozását az **eseménykezelő modul** végzi (5. sor). Ekkor az esemény alapján módosításra kerül(het) a példa, így a megoldása (azaz az aktuális szolgáltatási terv) is. A modul kimenete az aktualizált I példa és S megoldás mellett két, újratervezést megkönnyítő elem: az esemény által érintett feladat(ok) (T_k), a megoldás ezen feladat(ok) nélkül ($\overline{S}$), valamint egy logikai érték (*reroute*), ami megmondja, hogy van-e szükség újratervezésre. (Annak ellenére, hogy S egy megvalósítható megoldás, szükség lehet újratervezésre, mivel S nem feltétlenül optimális.) Ha szükség van újratervezésre, akkor az **újratervező modul** meghívásra kerül, ami megpróbálja újratervezni a szolgáltatási tervet (7. sor). A modul bemenetei az eseménykezelő modul kimenetei és a kimenetei az I példa és a korábbi korrigált megoldással egyező minőségű vagy annál jobb S megoldás. A példát csak akkor módosítja, ha a korábbi (esemény bekövetkezése előtti) megoldáshoz képest nőtt az útvonaltervek száma, mivel ekkor az új útvonaltervhez azonosítót és (ha van szabad jármű) járművet kell rendelni.

A szimulációnak akkor van vége, ha minden (működő) jármű visszatért a telephelyre (azaz a szabad és a meghibásodott járművek száma megegyezik az összes jármű számával) és nincs több végrehajtandó feladat (8. sor).

Az esemény létrehozó modul a D.5.2. alszakaszban, az eseménykezelő modul a D.5.3. alszakaszban, az újratervező modul pedig a D.5.4. alszakaszban kerül bemutatásra.

D.5.2. Az esemény létrehozó modul

Az adat-vezérelt DCARP keretrendszer teszteléséhez a szolgáltatási folyamat végrehajtását leíró események (utazási vagy szolgáltatási események) és váratlanul bekövetkező események (úthálózathoz vagy járműhöz kapcsolódó események) generálására van szükség. Mivel a váratlan események közül csak három számít kritikusnak (a „feladat megjelenés", az „igény növekedés" és a „jármű meghibásodás" esemény) – azaz csak azon események bekövetkezése esetén lehet nélkülözhetetlenül szükség újratervezésre, azaz DCARP megoldó használatára – ezért csak azokhoz készítettem eseménygenerátort. Ettől függetlenül minden eseménytípus kezelésére fel van készítve a keretrendszer, így bármilyen eseményt képes kezelni.

Mivel a kezdeti DCARP példa egy statikus CARP példa, a kezdeti megoldás egy statikus CARP megoldó segítségével kapható meg. Az esemény létrehozó modul az aktuális DCARP példa és szolgáltatási terv alapján próbál véletlenszerűen eseményeket létrehozni. A lehetséges események részletes elemzése a D.1. szakaszban olvasható, a keretrendszer által használt eseményfolyam felépítése pedig a 3.2.5. alszakaszban kerül bemutatásra.

Utazási vagy szolgáltatási esemény generálása

Egy utazási vagy szolgáltatási esemény négy attribútum értékből áll: időbélyeg (ts), az esemény által érintett ív (a), az esemény által érintett jármű azonosítója (h), valamint az esemény típusát jelző bit érték (α).

Az algoritmus az utazási és a szolgáltatási eseményeket az adott S szolgáltatási terv végrehajtásának szimulálásával generálja. Mivel az útvonaltervek csak a feladatíveket tartalmazzák, a szimulációhoz kiegészíti azokat a csak áthaladásra használt ívekkel is. Feltételezve, hogy az ívek áthaladási költségét és kiszolgálási költségét az áthaladásukra, illetve a kiszolgálásukra fordított idő határozza meg, az eddigi összköltséget használja az események időpontjaként (azaz ts értékeként). Az esemény generálása csak akkor történik meg, ha van még úton lévő működő jármű (azaz a szabad és a meghibásodott járművek száma kisebb, mint a járművek teljes létszáma; $|H_f| + |H_e| < |H|$). Az esemény generálása a következő képpen történik:

1. Mindegyik (működő) jármű $k \in R \setminus R_e$ útvonaltervére kiszámolja az eddigi teljes költség és a következő ív költségének az összegét. Ha van virtuális feladat (azaz $rt(k) \neq -1$), akkor annak a kiszolgálási költsége (azaz $sc(rt(k))$) megadja az eddigi költséget. Ha a következő $a \in A$ ív feladatív, akkor az ív szolgáltatási költségét (azaz $sc(a)$-t) adja hozzá, különben az ív áthaladási költségét (azaz $dc(a)$-t).

2. Az algoritmus annak az útvonaltervnek a következő $a \in A$ ívét választja, aminél a legkisebb a kiszámolt költség, mivel azon az íven történő áthaladás (illetve feladatív esetén kiszolgálás) fog legkorábban megtörténni. Ekkor a ts értékéhez hozzáadódik az ív áthaladási/szolgáltatási költsége. A h értéke az érintett útvonaltervet követő jármű azonosítója lesz. Az α értéke 1, ha az ív feladatív (azaz $a \in T$), különben az értéke 0 lesz.

„Feladat megjelenés” esemény generálása

Egy „feladat megjelenés” esemény négy attribútum értékből áll: időbélyeg (ts), az esemény által érintett ív (a), annak kiszolgálási költsége (sc_a) és igénye (dem_a).

Az algoritmus az aktuális I DCARP példából csak olyan $a \in A$ ívet választhat ki, amihez az adott pillanatban nem tartozik egyetlen egy feladat sem (azaz $a \notin T$). Ha nincs ilyen ív, akkor nem tud ilyen eseményt generálni. Ez nem zárja ki azt a lehetőséget, hogy olyan ív kerüljön kiválasztásra, ami korábban már kiszolgálásra került. A kiszolgálási költség értéke az egyszerűség kedvéért ugyanaz lesz, mint az áthaladási költségé (azaz $sc_a = dc(a)$). Az igény (dem_a) értéke egy véletlenszerűen választott 1 és q (a jármű maximális kapacitása) közötti egész szám.

„Igény növekedés” esemény generálása

Egy „igény növekedés” esemény (egy „feladat megjelenés” eseményhez hasonlóan) négy attribútum értékből áll: időbélyeg (ts), az esemény által érintett ív (a), annak kiszolgálási költsége (sc_a) és igénye (dem_a).

Az algoritmus az aktuális I DCARP példából csak olyan $a \in A$ ívet választhat ki, amihez az adott pillanatban már tartozik feladat, de az nem virtuális feladat (azaz $a \in T \setminus T_v$). Ha nincs ilyen ív, akkor nem tud ilyen eseményt generálni. Az új kiszolgálási költség és az igény értékének a meghatározása egy véletlenszerűen választott, 1 és $\frac{q}{dem(a)}$ közötti szorzóérték (inc_a) használatával történik. A megszabott értékkorlát biztosítja, hogy mindenképpen legyen növekedés, de az ne legyen akkora, hogy kiszolgálhatatlanná váljon a feladat. Az új kiszolgálási költség és igény értékét tehát úgy határozza meg, hogy a régi értékeket az inc_a értékével megszorozza (azaz $sc_a = sc(a) * inc_a$ és $dem_a = dem(a) * inc_a$).

„Jármű meghibásodás” esemény generálása

Egy „jármű meghibásodás” esemény három attribútum értékből áll: időbélyeg (ts), az esemény által érintett jármű azonosítója (h) és a jármű állapotváltozását jelölő érték (h_e), aminek az értéke meghibásodás esetén 1.

Az algoritmus az aktuális I DCARP példából csak olyan működő jármű h azonosítóját választhatja ki, ami éppen úton van (azaz $h \in H \setminus (H_f \cup H_e)$). Ha nincs ilyen jármű, akkor nem tud ilyen eseményt generálni.

D.5.3. Az eseménykezelő modul

Ebben az alszakaszban az eseménykezelő modult megvalósító algoritmus (18. algoritmus) kerül bemutatásra.

Az algoritmus bemenetei a legutóbbi I DCARP példa, annak S megoldása, valamint a megfigyelt e esemény. Az e esemény szolgáltatási folyamat végrehajtását leíró esemény (utazási vagy szolgáltatási esemény) és váratlanul bekövetkező esemény (úthálózathoz vagy járműhöz kapcsolódó esemény) is lehet. Az algoritmus a futása végén az aktualizált I példát, annak egy lehetséges S megoldását, valamint az újratervezéshez kapcsolódó elemeket ad át a fő algoritmusnak: az S megoldás csonka (T_k nélküli) változatát ($\overline{S}$), a csonka megoldásba beszúrandó feladat(ok) sorozatát (T_k), valamint az újratervezés szükségességét jelző logikai értéket ($reroute$).

Algoritmus 18: *eventHandlerModule* (Eseménykezelő modul)

input : $I = (V, v_0, A, T, T_v, n, w, q, H, H_e, H_f, R, R_e, rt, rv, head, tail, dc, inv, dem, sc, mdc)$,
$S = \{r_1, r_2, \ldots, r_K\}$,
$e \in \mathbb{Z}_{>0} \times A \cup \{-1\} \times H \cup \{-1\} \times \{-1, 0, 1\} \times \mathbb{R}_{\geq 0} \cup \{-1\} \times \mathbb{R}_{\geq 0} \cup \{-1\} \times \{-1, 0, 1\}$

1 **begin**
2 $ts \leftarrow \pi_1(e)$; // az időbélyeg kinyerése az eseményből
3 $a \leftarrow \pi_2(e)$; // az esemény által érintett útszakasz (ív) kinyerése az eseményből
4 $h \leftarrow \pi_3(e)$; // az esemény által érintett jármű azonosítójának kinyerése az eseményből
5 $\overline{S} \leftarrow S$; // a csonka megoldásjelölt inicializálása
6 $T_k \leftarrow \langle\rangle$; // a csonka megoldásjelöltbe beszúrandó feladatsorozat inicializálása
7 $reroute \leftarrow false$; // az újratervezés szükségességét jelző érték inicializálása
8 **if** $a \neq -1 \wedge h \neq -1$ **then** // **utazási vagy szolgáltatási esemény**
9 $\alpha \leftarrow \pi_4(e)$; // az esemény típusának (utazási vagy szolgáltatási) kinyerése az eseményből
10 $I, S \leftarrow travelServiceEventHandler(I, S, a, h, \alpha)$; // az érintett virtuális feladat aktualizálása
11 **else if** $a \neq -1$ **then** // **úthálózathoz kapcsolódó esemény**
12 $dc_a \leftarrow \pi_5(e)$; // az a ív új áthaladási költségének kinyerése az eseményből
13 $sc_a \leftarrow \pi_6(e)$; // az a ív (új) szolgáltatási költségének kinyerése az eseményből
14 $dem_a \leftarrow \pi_7(e)$; // az a ív (új) igényének kinyerése az eseményből
15 **if** $a \in T \wedge dem_a \neq -1$ **then** // ha az a ív nem új feladat és az igénye változott
16 $\overline{dem}_a \leftarrow dem(a)$; // az a ív legutóbbi igényének elmentése
17 $I \leftarrow updateArcValues(I, a, dc_a, sc_a, dem_a)$; // az értékek frissítése a példában
18 **if** $dc_a = \infty$ **then** // **„útlezárás" esemény**
19 $mdc \leftarrow recalculateMDC(I, a)$; // mdc frissítése az a-t tartalmazó útvonalak esetében
20 **if** $a \in T \wedge dc_a = -1 \wedge sc_a = -1 \wedge dem_a = -1$ **then** // **„feladat lemondás" esemény**
21 $I, S \leftarrow handleTaskCancellationEvent(I, S, a)$;
22 **if** $sc_a \neq -1 \wedge dem_a \neq -1$ **then**
23 **if** $a \notin T$ **then** // **„feladat megjelenés" esemény**
24 $I, S, \overline{S}, T_k \leftarrow handleTaskAppearanceEvent(I, S, a)$;
25 $reroute \leftarrow true$;
26 **else if** $\overline{dem}_a < dem_a$ **then** // **„igény növekedés" esemény**
27 $S, \overline{S}, T_k, reroute \leftarrow handleDemandIncreasedEvent(I, S, a)$;
28 **else if** $h \neq -1$ **then** // **járműhöz kapcsolódó esemény**
29 $h_e \leftarrow \pi_8(e)$; // a h jármű állapotának kinyerése az eseményből
30 **if** $h_e = 0$ **then** // **„jármű beüzemelés" esemény**
31 $H_e \leftarrow H_e \setminus \{h\}$; // a h jármű eltávolítása a hibás járművek halmazából
32 $k \leftarrow \max\{x | x \in R, rv(x) = h\}$; // a h jármű legutóbbi útvonaltervének megkeresése
33 $R_e \leftarrow R_e \setminus \{k\}$; // a k útvonalterv eltávolítása a felfüggesztett útvonaltervek halmazából
34 **else if** $h_e = 1$ **then** // **„jármű meghibásodás" esemény**
35 $I, S, \overline{S}, T_k, reroute \leftarrow handleVehicleBreakdownEvent(I, S, h)$;
36 **return** $I, S, \overline{S}, T_k, reroute$;

Az algoritmus először kinyeri az eseményből a ts időbélyeget, az esemény által érintett a útszakaszt (ívet), valamint az esemény által érintett jármű h azonosítóját (2–4. sor), majd inicializálja az $\overline{S}$ csonka megoldásjelöltet, a csonka megoldásba beillesztendő T_k feladato(ka)t tartalmazó feladatsorozatot, illetve az újratervezés szükségességét jelző logikai változó ($reroute$) értékét (5–7. sor).

Ha az útszakasz (a) és a jármű azonosítója (h) is adott (azaz az értékük nem -1), akkor az e egy **utazási vagy szolgáltatási esemény** (8. sor). Ekkor az algoritmus még pluszba kinyeri az eseményből annak típusát (α), ami azt határozza meg, hogy utazási vagy szolgáltatási eseményről van-e szó (9. sor), majd annak függvényében az esemény által érintett virtuális feladatot aktualizálja a $travelServiceEventHandler$ eseménykezelő függvény használatával (10. sor). Ekkor az I példa és annak S megoldása is módosul, mivel a példában és a megoldásban is frissíteni kell az adott virtuális feladatot. Az utazási és a szolgáltatási esemény kezelését (azaz az esemény által érintett virtuális feladat aktualizálását) a 19. algoritmus írja le részletesen.

Ha csak az útszakasz (a) adott, akkor az e egy **úthálózathoz kapcsolódó esemény** (11. sor). Ekkor az algoritmus még pluszba kinyeri az eseményből az a ívhez tartozó (új) áthaladási költséget (dc_a), szolgáltatási költséget (sc_a), valamint igényt (dem_a) (12–14. sor). Ha az a nem új feladat és az igénye változott (azaz az ese-

ményből kinyert igény értéke nem -1), akkor az a legutóbbi igény értéke elmentésre kerül a $\overline{dem}_a$ változóba (15–16. sor). Ezt követően az I példában frissítésre kerülnek az érintett értékek az $updateArcValues$ függvénnyel (17. sor). Természetesen csak a megadott (azaz nem -1) értékek kerülnek rögzítésre. Ha az a egy új feladat, akkor ez a függvény végzi az a hozzáadását az sc és a dem függvények értelmezési tartományához, illetve az sc_a és a dem_a értékek hozzárendelését is. A feladat hozzáadását a $handleTaskAppearanceEvent$ függvény (azaz a „feladat megjelenés" eseménykezelő) intézi (24. sor).

Ha az útszakasz (a) új áthaladási költsége végtelen (∞), akkor az e egy **„útlezárás" esemény** (18. sor). Ekkor az algoritmus az a ívet tartalmazó útvonalak esetében újraszámolja a minimális teljes áthaladási költséget, azaz az mdc függvényt aktualizálja a $recalculateMDC$ függvény segítségével (19. sor). Ha az a ív egy feladat (azaz $a \in T$) és nincs hozzá a három említett érték közül egyik sem megadva (azaz mindegyiknek az eseményből kinyert értéke -1), akkor az e egy **„feladat lemondás" esemény** (20. sor). Ekkor a $handleTaskCancellationEvent$ függvény elvégzi az a feladat eltávolítását az I példából és annak S megoldásából is (21. sor). A függvény működését a 20. algoritmus írja le részletesen. Ha az a ívhez (új) kiszolgálási költség és igény is meg van adva (azaz sc_a és dem_a sem -1), akkor új feladat jelent meg (22. sor). Ha az a ív még nem szerepel a kiszolgálandó feladatok halmazában, akkor e egy **„feladat megjelenés" esemény** (23. sor). Ez azt jelenti, hogy egy olyan íven jelent meg új kiszolgálandó feladat, ahol a jelenlegi I példa szerint nincs egy sem. Az ilyen eseményt a $handleTaskAppearanceEvent$ függvény dolgozza fel (24. sor). A függvény működését a 21. algoritmus írja le részletesen. Ilyenkor (a függvény kimeneteitől függetlenül) mindig szükség van újratervezésre, ezért a $reroute$ változó értéke igaz lesz (25. sor). Ha az a ív már szerepel a kiszolgálandó feladatok halmazában és az új igénye (dem_a) nagyobb, mint a korábbi ($\overline{dem}_a$), akkor e egy **„igény növekedés" esemény** (26. sor). Ez azt jelenti, hogy egy olyan íven jelent meg új kiszolgálandó feladat, ahol előtte is volt. Az ilyen eseményt a $handleDemandIncreasedEvent$ függvény dolgozza fel (27. sor). A függvény működését a 22. algoritmus írja le részletesen.

Ha csak a jármű azonosítója (h) adott, akkor az e egy **járműhöz kapcsolódó esemény** (28. sor). Ekkor az algoritmus még pluszba kinyeri az eseményből a h azonosítójú jármű állapotát jelző h_e bit értéket (29. sor). Ha a h_e értéke 0, akkor **„jármű beüzemelés" esemény**ről van szó (30. sor). Ekkor – feltételezve, hogy csak egy korábban meghibásodott jármű kerülhet beüzemelésre – a jármű h azonosítóját eltávolítja a hibás járművek halmazából (31. sor). Ezt követően megkeresi a jármű legutóbb követett útvonaltervének az azonosítóját (itt k-t) (32. sor), majd eltávolítja azt a felfüggesztett útvonaltervek halmazából (33. sor). Ha a h_e értéke 1, akkor **„jármű meghibásodás" esemény**ről van szó, amit a $handleVehicleBreakdownEvent$ eseménykezelő függvény dolgoz fel (34–35. sor). A függvény működését a 23. algoritmus írja le részletesen.

Az utazási és a szolgáltatási esemény kezelése

Ebben az alszakaszban az utazási és a szolgáltatási eseménykezelő függvény működését leíró algoritmus (19. algoritmus) kerül bemutatásra.

Az algoritmus bemenetei a legutóbbi I DCARP példa, annak S megoldása, valamint a megfigyelt e eseményből kinyert $a \in A$ ív, $h \in H$ jármű azonosító és α tevékenységtípus jelző bit érték. Az algoritmus célja a h azonosítójú jármű virtuális

feladatának aktualizálása az a íven (útszakaszon) végrehajtott (az α által meghatározott) utazási vagy szolgáltatási esemény alapján. Az algoritmus a futása végén az aktualizált I példát és annak S megoldását adja vissza.

Algoritmus 19: *travelServiceEventHandler* (Utazási és szolgáltatási esemény kezelése)

```
   input  : I = (V, v_0, A, T, T_v, n, w, q, H, H_e, H_f, R, R_e, rt, rv, head, tail, dc, inv, dem, sc, mdc),
            S = {r_1, r_2, ..., r_K}, a ∈ A, h ∈ H, α ∈ {0, 1}
1  begin
2      k ← max{x|x ∈ R, rv(x) = h};                          // a h jármű aktuális útvonaltervének megkeresése
3      if tail(a) = v_0 then                                  // ha a h jármű visszatért a telephelyre
4          R_e ← R_e ∪ {k};                                   // a h jármű k útvonaltervének végrehajtásának felfüggesztése
5          if ∃x ∈ R s.t. rv(x) = −1 then                     // ha van még szabad útvonalterv
6              x ← min{x|x ∈ R, rv(x) = −1};                  // a következő útvonalterv megkeresése
7              rv(x) ← h;                                     // az útvonalterv hozzárendelése a h járműhöz
8          else
9              H_f ← H_f ∪ {h};                               // a h jármű felszabadítása
10     vt ← (v_0, tail(a));                                   // új virtuális feladat (vt) létrehozása
11     A ← A ∪ {vt};                                          // vt hozzáadása az ívek halmazához
12     T ← T ∪ {vt};                                          // vt hozzáadása a feladatok halmazához
13     T_v ← T_v ∪ {vt};                                      // vt hozzáadása a virtuális feladatok halmazához
14     head ← head ∪ {vt → v_0};                              // vt fejcsúcsának meghatározása
15     tail ← tail ∪ {vt → tail(a)};                          // vt végcsúcsának meghatározása
16     dc ← dc ∪ {vt → ∞};                                    // vt áthaladási költségének meghatározása
17     if α = 0 then                                          // utazási esemény
18         sc ← sc ∪ {vt → dc(a)};                            // vt kiszolgálási költségének meghatározása
19         dem ← dem ∪ {vt → 0};                              // vt igényének meghatározása
20     else if α = 1 then                                     // szolgáltatási esemény
21         sc ← sc ∪ {vt → sc(a)};                            // vt kiszolgálási költségének meghatározása
22         dem ← dem ∪ {vt → dem(a)};                         // vt igényének meghatározása
23         I, S ← handleTaskCancellationEvent(I, S, a);       // az elvégzett a feladat eltávolítása
24     if rt(k) ≠ −1 then                                     // ha a k útvonaltervhez már létezik virtuális feladat
25         sc(vt) ← sc(rt(k)) + sc(vt);                       // vt kiszolgálási költségének kiegészítése a korábbi értékével
26         dem(vt) ← dem(rt(k)) + dem(vt);                    // vt igényének kiegészítése a korábbi értékével
27         I, S ← removeVirtualTask(I, S, rt(k));             // a régi virtuális feladat eltávolítása
28     rt(k) ← vt;                                            // az új virtuális feladat (vt) hozzárendelése az útvonaltervhez
29     S ← insertTask(S, k, 1, vt);                           // vt beszúrása a megoldásba
30     return I, S;
```

Az algoritmus először megkeresi a h azonosítójú jármű aktuális k azonosítójú útvonaltervét (2. sor), majd ellenőrzi, hogy a jármű befejezte-e az útvonalterv végrehajtását (azaz elérte-e a v_0 telephelyet) (3. sor). Ha igen, akkor hozzáadja a k útvonalterv azonosítóját azon útvonaltervek azonosítóinak halmazához, amelyeknek a végrehajtása leállt (R_e) (4. sor). Ha van még olyan x útvonalterv, amihez nincs jármű rendelve (azaz $rv(x) = -1$) (5. sor), akkor megkeresi a legrégebbi ilyen útvonaltervet (6. sor), majd hozzárendeli a h azonosítójú járművet (7. sor). Ha nincs ilyen útvonalterv, akkor a járművet felszabadítja, azaz a h-t hozzáadja a (jelenleg) szabad járművek azonosítóinak halmazához (H_f) (9. sor).

Az algoritmus ezután létrehozza az új virtuális feladatot (10. sor), majd hozzáadja azt az ívek halmazához (11. sor), az elvégzendő feladatok halmazához (12. sor) és a virtuális feladatok halmazához (13. sor). Ezután létrehozza hozzá az ívet jellemző *head*, *tail* és *dc* függvény értékeket (14–16. sor). Utazási esemény esetén (azaz, ha az α értéke 0) (17. sor) az új virtuális feladat kiszolgálási költség értéke (sc) az érintett ív áthaladási költség értékével (azaz $dc(a)$-val) nő (18. sor), az igénye (dem) pedig nem nő, ezért az értéke 0 lesz (19. sor). Szolgáltatási esemény esetén (azaz, ha az α értéke 1) (20. sor) az új virtuális feladat kiszolgálási költség értéke (sc) és igénye (dem) az elvégzett a feladat kiszolgálási költség értékével ($sc(a)$) és igényével ($dem(a)$) nő (21–22. sor). Továbbá, az elvégzett a feladatot eltávolítja az I példából és annak S megoldásából (23. sor). Mivel ennek a menete megegyezik a „feladat lemondás”

esemény kezelésével, ezért ehhez az azt megvalósító $handleTaskCancellationEvent$ függvényt használja (20. algoritmus).

Ha a k azonosítójú útvonaltervhez már létezik virtuális feladat (24. sor), akkor az ahhoz tartozó kiszolgálási költség értéket és igényt hozzáadja az új virtuális feladatához (25–26. sor). Ezt követően eltávolítja a régi virtuális feladatot az I példából és annak S megoldásából is a $removeVirtualTask$ függvény használatával (27. sor). Ez azt jelenti, hogy a virtuális feladat eltávolításra kerül az ívek halmazából (azaz A halmazból) és a feladat halmazokból (azaz T és T_v halmazokból) is, így a $head$, a $tail$, a dc, az sc és a dem értékei is törlődnek.

Az algoritmus legvégül hozzárendeli az új virtuális feladatot (vt-t) a k azonosítójú útvonaltervhez (28. sor), majd beszúrja azt az S megoldásba (29. sor).

A „feladat lemondás" esemény kezelése

Ebben az alszakaszban a „feladat lemondás" eseménykezelő függvény működését leíró algoritmus (20. algoritmus) kerül bemutatásra.

Algoritmus 20: $handleTaskCancellationEvent$ („Feladat lemondás" esemény kezelése)

```
   input : I = (V, v_0, A, T, T_v, n, w, q, H, H_e, H_f, R, R_e, rt, rv, head, tail, dc, inv, dem, sc, mdc),
           S = {r_1, r_2, ..., r_K}, a ∈ A
 1 begin
 2     S ← removeTask(S, a);                          // az a feladat eltávolítása a megoldásból
 3     n ← n − 1;                                     // a kiszolgálandó feladatok számának csökkentése eggyel
 4     T ← T \ {a};                                   // az a feladat eltávolítása a feladathalmazból
 5     sc ← sc \ {a → sc(a)};                         // az a feladat szolgáltatási költségének eltávolítása
 6     dem ← dem \ {a → dem(a)};                      // az a feladat igényének eltávolítása
 7     if a ∈ dom(inv) then                           // ha az a feladat élfeladat volt
 8         T ← T \ {inv(a)};                          // az inv(a) feladat eltávolítása a feladathalmazból
 9         sc ← sc \ {a → sc(inv(a))};                // az inv(a) feladat szolgáltatási költségének eltávolítása
10         dem ← dem \ {a → dem(inv(a))};             // az inv(a) feladat igényének eltávolítása
11     return I, S;
```

Az algoritmus bemenetei a legutóbbi I DCARP példa, annak S megoldása, valamint a megfigyelt e eseményből kinyert $a \in T$ ívfeladat. Az algoritmus a futása végén az aktualizált I példát és annak S megoldását adja vissza, amikben már nem szerepel az a feladat és a hozzá kapcsolódó értékek.

Az algoritmus először az a feladatot eltávolítja az S megoldásból (2. sor) és a kiszolgálandó feladatok számát (n) eggyel csökkenti (3. sor). Ezt követően az a feladatot a T feladathalmazból is eltávolítja, valamint a hozzá kapcsolódó szolgáltatási költséget (sc) és igényt (dem) is törli (4–6. sor). Ha az a feladat élfeladat volt, akkor ugyanezeket az $inv(a)$ feladatra is elvégzi (7–10. sor).

A „feladat megjelenés" esemény kezelése

Ebben az alszakaszban a „feladat megjelenés" eseménykezelő függvény működését leíró algoritmus (21. algoritmus) kerül bemutatásra.

Az algoritmus bemenetei a legutóbbi I DCARP példa, annak S megoldása, valamint a megfigyelt e eseményből kinyert $a \in A$ ív. Az algoritmus hozzáadja az I példához az új feladatot, majd a futása végén az aktualizált I példát, annak egy lehetséges S megoldását, az S megoldás csonka (a nélküli) változatát ($\overline{S}$), valamint a csonka megoldásba beszúrandó feladatok sorozatát (T_k-t, ami a-t tartalmazza) adja vissza. Az új feladathoz tartozó szolgáltatási költséget és igényt az $updateArcValues$

függvény addja hozzá a példához még az eseménykezelő függvény meghívása előtt az eseménykezelő modulban (18. algoritmus, 17. sor).

Algoritmus 21: *handleTaskAppearanceEvent* („Feladat megjelenés” esemény kezelése)

```
   input : I = (V, v_0, A, T, T_v, n, w, q, H, H_e, H_f, R, R_e, rt, rv, head, tail, dc, inv, dem, sc, mdc),
           S = {r_1, r_2, ..., r_K}, a ∈ A
1  begin
2      n ← n + 1;                                   // a kiszolgálandó feladatok számának növelése eggyel
3      T ← T ∪ {a};                                 // a hozzáadása a feladathalmazhoz
4      if a ∈ dom(inv) then                         // ha az a ív élből származik
5          T ← T ∪ {inv(a)};                        // inv(a) hozzáadása a feladathalmazhoz
6      S̄ ← S;                                       // csonka megoldásjelölt létrehozása
7      S ← S̄ ∪ {⟨t_0, a, t_0⟩};                     // megoldásjelölt létrehozása (a hozzáadása új útvonaltervben)
8      T_k ← ⟨a⟩;   // a csonka megoldásba beillesztendő feladatot tartalmazó feladatsorozat létrehozása
9      return I, S, S̄, T_k;
```

Az algoritmus először a kiszolgálandó feladatok számát (n) eggyel növeli (2. sor), majd hozzáadja az a ívet a kiszolgálandó feladatok T halmazához (3. sor). Ha az a ív élből származik, akkor az $inv(a)$ ívet is hozzáadja a T feladathalmazhoz (4–5. sor). Ezt követően létrehozza az $\overline{S}$ csonka megoldásjelöltet, ami ugyanaz, mint a korábbi S megoldás (6. sor). Az aktualizált S megoldást úgy hozza lére, hogy az új a feladatot hozzáadja egy új útvonaltervhez, amit hozzáfűz az $\overline{S}$ csonka megoldás végéhez (7. sor). Legvégül létrehozza a csonka megoldásba beillesztendő a feladatot tartalmazó T_k feladatsorozatot (8. sor).

Az „igény növekedés” esemény kezelése

Ebben az alszakaszban az „igény növekedés” eseménykezelő függvény működését leíró algoritmus (22. algoritmus) kerül bemutatásra.

Algoritmus 22: *handleDemandIncreasedEvent* („Igény növekedés” esemény kezelése)

```
   input : I = (V, v_0, A, T, T_v, n, w, q, H, H_e, H_f, R, R_e, rt, rv, head, tail, dc, inv, dem, sc, mdc),
           S = {r_1, r_2, ..., r_K}, a ∈ A
1  begin
2      i ← findTaskRoutePlanIndex(S, a);  // az a-t tartalmazó útvonalterv azonosítójának megkeresése
3      load_i = load(I, r_i);                          // az r_i útvonaltervre vonatkozó terhelés kiszámítása
4      if load_i > q then                  // ha a terhelés nagyobb, mint a jármű maximális kapacitása
5          S̄ ← removeTask(S, a);      // csonka megoldásjelölt létrehozása (a eltávolítása a megoldásból)
6          S ← S̄ ∪ {⟨t_0, a, t_0⟩};                 // megoldásjelölt létrehozása (a hozzáadása új útvonaltervben)
7          T_k ← ⟨a⟩; // a csonka megoldásba beillesztendő feladatot tartalmazó feladatsorozat létrehozása
8          return S, S̄, T_k, true;
9      return S, S, ∅, false;
```

Az algoritmus bemenetei a legutóbbi I DCARP példa, annak S megoldása, valamint a megfigyelt e eseményből kinyert $a \in A$ ív. Az algoritmus megnézi, hogy az igénynövekedést követően az érintett járműnek még mindig van-e elég kapacitása az útvonaltervében szereplő összes feladat elvégzésre vagy sem. Ha nem, akkor az a feladatot kiveszi az útvonaltervből és egy újba rakja azt, így létrehozva az új S megoldást. A futása végén az aktualizált I példát, a létrehozott S megoldást, az S megoldás csonka (a nélküli) változatát ($\overline{S}$), a csonka megoldásba beszúrandó feladatok sorozatát (T_k-t, ami a-t tartalmazza), valamint az újratervezés szükségességét jelző logikai igaz értéket adja vissza. Ha az igény növekedés nem befolyásolja az

útvonalterv megvalósíthatóságát, akkor az S megoldás változatlan marad, a T_k-ra nincs szükség (helyette üreshalmaz kerül visszaadásra) és így újratervezésre sincs szükség (azaz a *reroute* értéke hamis lesz).

Az algoritmus először megkeresi az a feladatot tartalmazó útvonalterv i azonosítóját (2. sor), majd kiszámolja az r_i útvonaltervre vonatkozó jelenlegi $load_i$ terhelést a *load* függvénnyel (3. sor). Ha a terhelés nagyobb, mint a jármű maximális kapacitása (4. sor), akkor az a feladatot eltávolítja az S megoldásból, így létrehozva az $\overline{S}$ csonka megoldásjelöltet (5. sor). Az S megoldásjelölt létrehozásához új útvonaltervbe illeszti az a feladatot (6. sor), majd legvégül létrehozza a csonka megoldásba beillesztendő a feladatot tartalmazó T_k feladatsorozatot (7. sor).

A „jármű meghibásodás" esemény kezelése

Ebben az alszakaszban a „jármű meghibásodás" eseménykezelő függvény működését leíró algoritmus (23. algoritmus) kerül bemutatásra.

Algoritmus 23: *handleVehicleBreakdownEvent* („Jármű meghibásodás" esemény kezelése)

```
   input : I = (V, v_0, A, T, T_v, n, w, q, H, H_e, H_f, R, R_e, rt, rv, head, tail, dc, inv, dem, sc, mdc),
           S = {r_1, r_2, ..., r_K}, h ∈ H
1  begin
2      H_e ← H_e ∪ {h};                          // a h jármű hozzáadása a hibás járművek halmazához
3      k ← max{x|x ∈ R, rv(x) = h};              // a h jármű aktuális útvonaltervének megkeresése
4      R_e ← R_e ∪ {k};                          // a k útvonalterv hozzáadása a felfüggesztett útvonaltervek halmazához
5      T_k ← getTasks(r_k);                      // a k útvonaltervben szereplő kiszolgálandó feladatok kimentése
6      if |T_k| > 0 then                         // ha van még kiszolgálandó feladat
7          S̄ ← removeTasks(S, T_k);              // csonka megoldásjelölt létrehozása
8          S ← S̄ ∪ {⟨t_0⟩ṫ_k⟨t_0⟩};               // megoldásjelölt létrehozása
9          return I, S, S̄, T_k, true;
10     return I, S, S, ∅, false;
```

Az algoritmus bemenetei a legutóbbi I DCARP példa, annak S megoldása, valamint a megfigyelt e eseményből kinyert $h \in H$ járműazonosító. Az algoritmus megnézi, hogy a meghibásodott jármű útvonalában szerepel-e még kiszolgálandó feladat (T_k). Ha igen, akkor a futása végén az aktualizált I példát, a létrehozott S megoldást, az S megoldás csonka (T_k nélküli) változatát ($\overline{S}$), a csonka megoldásba beszúrandó feladatok sorozatát (T_k), valamint az újratervezés szükségességét jelző logikai igaz értéket adja vissza. Ha nem, akkor csak az I példa változik, az S megoldás változatlan marad, a T_k-ra nincs szükség (helyette üreshalmaz kerül visszaadásra) és így újratervezésre sincs szükség (azaz a *reroute* értéke hamis lesz).

Az algoritmus először hozzáadja a $h \in H$ járműazonosítót a hibás járművek halmazához (azaz H_e-hez) (2. sor), majd megkeresi a jármű aktuális útvonaltervének $k \in R$ azonosítóját (3. sor) és azt hozzáadja a felfüggesztett útvonaltervek halmazához (azaz R_e-hez) (4. sor). Következő lépésként az r_k útvonaltervben szereplő kiszolgálandó feladatokat a T_k feladatsorozatba menti (5. sor). Ha a T_k mérete nagyobb mint 0 (azaz van még kiszolgálandó feladat) (6. sor), akkor szükség van újratervezésre. Ilyen esetben a T_k feladato(ka)t eltávolítja az S megoldásból, így létrehozva az $\overline{S}$ csonka megoldásjelöltet (7. sor). Az S megoldásjelölt létrehozásához új útvonaltervbe illeszti a T_k feladato(ka)t (8. sor).

D.5.4. Az újratervező modul

Ebben az alszakaszban az újratervező modult megvalósító algoritmus (24. algoritmus) kerül bemutatásra.

Algoritmus 24: *replanningModule* (Újratervező modul)

```
   input  : I = (V, v_0, A, T, T_v, n, w, q, H, H_e, H_f, R, R_e, rt, rv, head, tail, dc, inv, dem, sc, mdc),
            S = {r_1, r_2, ..., r_K, r_{K+1}}, S̄ = {r_1, r_2, ..., r_K}, T_k ∈ T*
1  begin
2      S ← RR1(I, S, S̄, T_k);                                   // az RR1 algoritmus futtatása
3      S ← CARPABC2(I, S);          // a feltárásra összpontosító CARP-ABC algoritmus futtatása
4      if |S| = |S̄| + 1 then                                    // ha van új útvonalterv
5          k ← max(R) + 1;                                      // a k útvonalterv azonosító létrehozása
6          R ← R ∪ {k};                                         // a k hozzáadása az útvonaltervek halmazához
7          rt ← rt ∪ {k → −1}; // a k hozzáadása a virtuális feladat függvény értelmezési tartományához
8          rv ← rv ∪ {k → −1};          // a k hozzáadása a jármű függvény értelmezési tartományához
9          if H_f ≠ ∅ then                                      // ha van szabad jármű
10             h ← randomChoice(H_f);                           // tetszőleges h jármű azonosító kiválasztása
11             rv(k) ← h;                                       // a h jármű hozzárendelése a k útvonaltervhez
12             H_f ← H_f \ {h};                                 // a h jármű eltávolítása a szabad járművek halmazából
13     return I, S;
```

Az algoritmus bemenetei az aktualizált I DCARP példa, annak egy lehetséges S megoldása, az S megoldás csonka (T_k nélküli) változata ($\overline{S}$), valamint a csonka megoldásba beszúrandó feladato(ka)t tartalmazó T_k feladatsorozat. Az algoritmus célja az S megoldásnál jobb (azaz költséghatékonyabb) megoldást találni. Az algoritmus a futása végén (a használt DCARP megoldó kimenete alapján) vagy a bemeneti S megoldást, vagy egy jobb megoldást ad vissza. Az I példán csak akkor módosít, ha a szolgáltatási tervek száma nőtt a korábbi (az esemény bekövetkezése előtt követett) szolgáltatási tervhez képest.

Az algoritmus először az RR1 újratervező algoritmust hívja meg (2. sor), majd a választott DCARP megoldó algoritmust (itt most a CARP-ABC algoritmust, amit az ABC függvény képvisel) (3. sor) a bemeneti S megoldás javítása érdekében. Ha az esemény bekövetkezését követően új útvonalterv bevonása szükséges a szolgáltatás elvégzéséhez (azaz az S megoldás elemeinek száma eggyel nagyobb mint az $\overline{S}$ csonka megoldásé), akkor szükség van az I példa módosítására (4. sor). Ilyen esetben az új útvonalterv k azonosítóját először létrehozza (5. sor), majd hozzáadja az összes útvonalterv azonosítójának R halmazához (6. sor). Az új útvonaltervhez tartozó virtuális feladatot, illetve az útvonaltervet végrehajtó jármű azonosítóját visszaadó függvényeket -1 értékkel inicializálja (7–8. sor). Ha van éppen szabad jármű (9. sor), akkor véletlenszerűen választ egyet (10. sor) és a h azonosítóját hozzárendeli az útvonalterv azonosítójához (11. sor). Mivel ezt követően az érintett jármű nem szabad többé, az azonosítóját kiveszi a szabad járművek halmazából (12. sor).

D.6. A hatékonyság vizsgálatokhoz felhasznált példák

CARP teljesítménymérő tesztkészletek

A CARP megoldók tesztelésére a szakirodalomban gyakran használnak teljesítménymérő tesztkészleteket. Ezek a tesztkészletek két fő kategóriába sorolhatók: szintetikus (pl. véletlenszerűen generált példákat tartalmazó) [153, 154, 155] és valós életen alapuló (azaz valós úthálózatokon és feladatokon alapuló példákat tartalmazó) [27, 126, 124]. Mivel egy algoritmus tesztelése az összes példán időigényes lenne, a CARP példákon végzett hatékonyság vizsgálathoz csak a következő öt példát választottam ki:

- „kshs1" a KSHS készletből [153];
- „egl-e1-A" és „egl-s1-A" az EGL készletből [27];
- „egl-g1-A" és „egl-g2-A" az EGL-Large készletből [126].

Az esemény generátort használó DCARP keretrendszerrel végzett hatékonyság vizsgálathoz csak az „egl-e1-A" példa került felhasználásra. A művelet hatékonyság vizsgálathoz az „egl-e1-A", „egl-s1-A" és „egl-g1-A" példák kerültek felhasználásra.

Az EGL és az EGL-Large készletek egy lancashire-i (egyesült királyságbeli) téli sószórási feladat adataiból származnak. Az EGL készlet 24 példát tartalmaz, amelyekben két különböző gráfot kombináltak különböző attribútumértékekkel. Azért választottam ki az „egl-e1-A" és az „egl-s1-A" példákat, hogy mindkét gráf képviselve legyen. Az EGL-Large készlet 10 példát tartalmaz, amelyekben a gráf ugyanaz, de a feladat élek száma 5 példában 347, a másik 5 példában pedig 375. Azért választottam ki az „egl-g1-A" és az „egl-g2-A" példákat, hogy mindkét példatípus képviselve legyen.

Az öt kiválasztott példa attribútumai a D.4. táblázatban röviden összegezve vannak. Ezeket a CARP példákat úgy választottam ki, hogy különböző méretű CARP-okat reprezentáljanak, így azok különböző bonyolultsági szintű megoldásokat igényelnek. Látható, hogy a „kshs1" példa egy kicsi szintetikus CARP példa, ami kis keresési teret jelent az algoritmusok számára, így a probléma nehézsége szempontjából a könnyű kategóriába sorolható. Az EGL példák valós életen alapuló CARP példák, így természetesen jóval bonyolultabbak. Méretük alapján az „egl-e1-A" példa nehézsége közepes, az „egl-s1-A" példa nehéz, az „egl-g1-A" és az „egl-g2-A" példák nehézsége pedig nagyon nehéz.

D.4. táblázat. A felhasznált CARP példák attribútumai

Attribútum	**kshs1**	**egl-e1-A**	**egl-s1-A**	**egl-g1-A**	**egl-g2-A**
a csúcsok száma	8	77	140	255	255
a feladat élek száma	15	51	75	347	375
az egyéb élek száma	0	47	115	28	0
a járművek száma	4	5	7	20	22
a járművek kapacitása	150	305	210	28 600	28 000
a teljes költség alsó korlátja	8 705	1 468	1 394	553 696	604 228
a legjobb megoldás költsége	14 661	3 548	5 018	992 045	1 088 040

DCARP forgatókönyvek

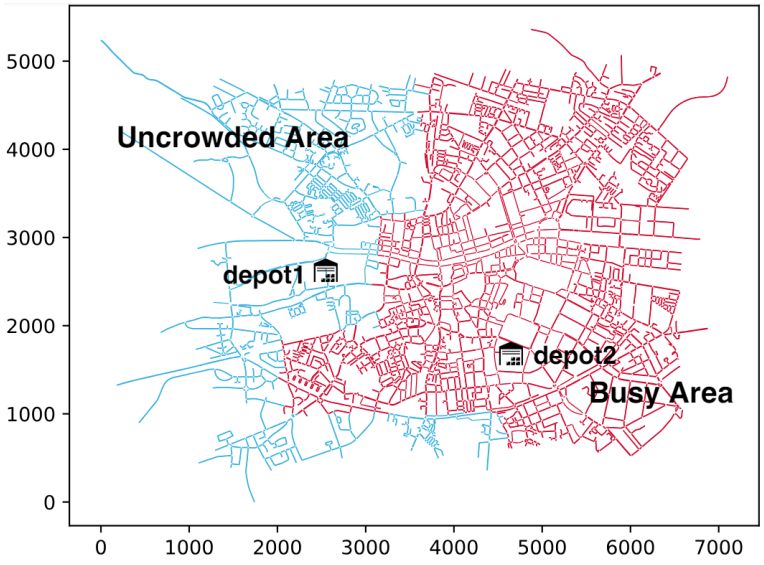
A forgalom szimulációt használó DCARP keretrendszer használatával végzett hatékonyság vizsgálatokhoz Dublin városának nyílt forráskódú, 24 órás forgalmi adatait [156] és az ezekből generált 12 DCARP forgatókönyvet [147] használtam fel. Pontosabban minden egyes forgatókönyvnek csak a kezdeti példáját használtam fel, mivel a rákövetkező példák az alkalmazott (D)CARP megoldó által visszaadott szolgáltatási terv függvényében változnak. Mindegyik forgatókönyvben ugyanaz az úthálózat, a feladatok száma, a járművek száma és a járműkapacitás:

- a csúcsok száma: 2 895;
- az ívek száma: 6 633;
- a feladatívek száma: 200;
- a járművek száma: ∞;
- a járművek kapacitása: 200.

A forgatókönyvek között a különbségek a szolgáltatás kezdési időpontjában, a telephely elhelyezkedésében és a feladatok eloszlásában vannak. Ezek a következők lehetnek:

- a szolgáltatás kezdési időpontja: csúcsidőn kívüli órák, csúcsidőhöz közeli órák;
- a telephely elhelyezkedése: belváros, külváros;
- a feladatok eloszlása: egész város, forgalmas terület, nem forgalmas terület.

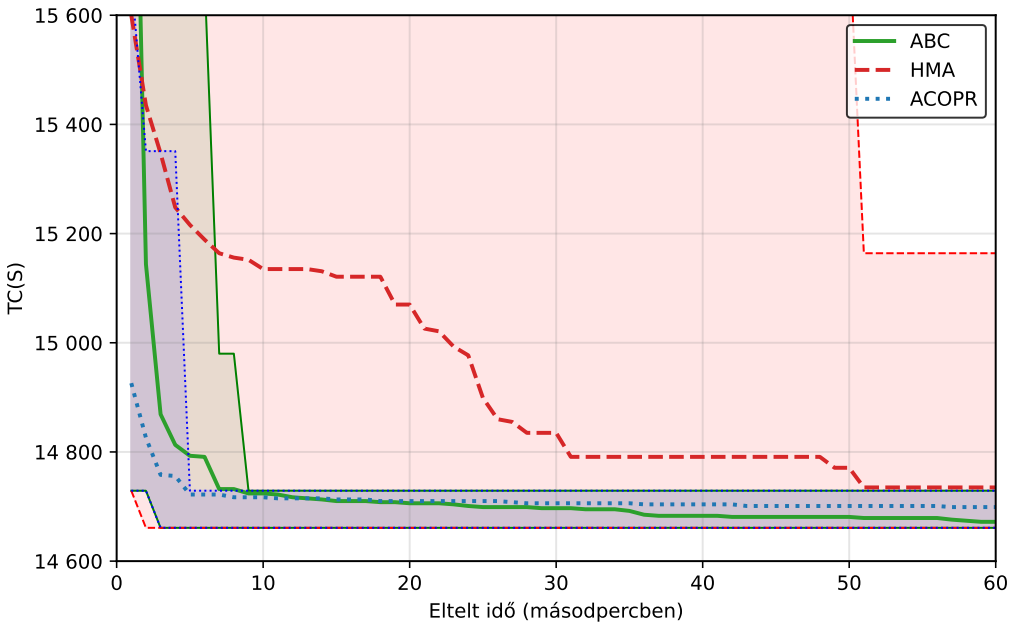
A D.1. ábrán a forgalmas terület („Busy Area”) és a nem forgalmas terület („Uncrowded Area”) eloszlása, valamint a telephely két lehetséges elhelyezkedése („depot1” és „depot2”) látható az úthálózatra vetítve.



D.1. ábra. Dublin úthálózata a forgalmas és a nem forgalmas terület eloszlásával, valamint a telephely két lehetséges elhelyezkedésével [147]

D.7. A javasolt CARP-ABC algoritmusok és az RR1 algoritmus hatékonyságának vizsgálatának kimenetei

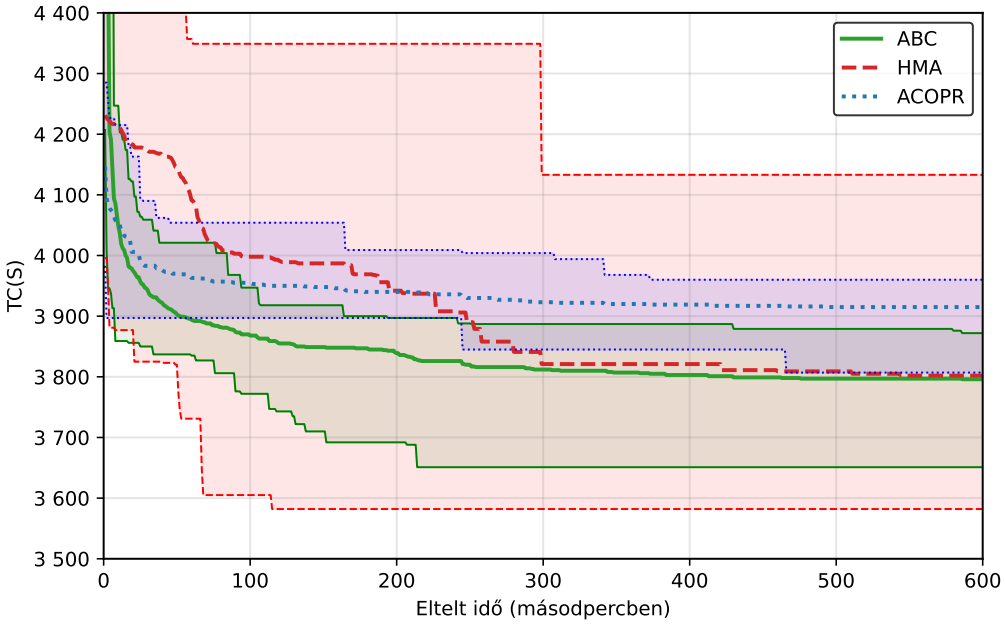
D.7.1. A felfedezésre összpontosító CARP-ABC algoritmus hatékonyságának vizsgálata CARP példákon



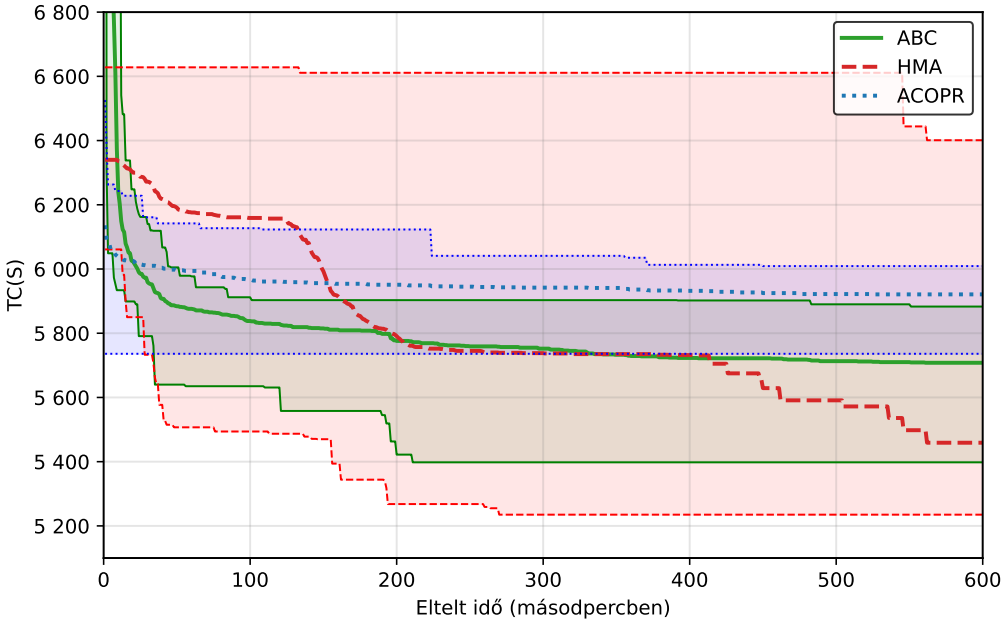
D.2. ábra. A HMA, az ACOPR és az ABC algoritmus 30 független futtatásának konvergencia sebessége a „kshs1” példán, egy diagramon ábrázolva

D.5. táblázat. A HMA és az ABC algoritmus globálisan legjobb megoldásának teljes költségének statisztikái az „egl-e1-A” példán, különböző időkorlátok között

Algoritmus	**Statisztikai mérőszám**	**Kimenetek különböző időkorlátok esetén**		
		$\leq$ 1 perc	**$\leq$ 5 perc**	**$\leq$ 10 perc**
ABC	Minimum	3 835	3 651	3 651
	Maximum	4 021	3 887	3 872
	Átlag	3 894,9	3 812,5	3 796,23
	Szórás	38,02	68,90	65,70
HMA	Minimum	3 731	3 582	3 582
	Maximum	4 357	4 133	4 133
	Átlag	4 091,6	3 821,5	3 802,4
	Szórás	184,92	140,69	148,10



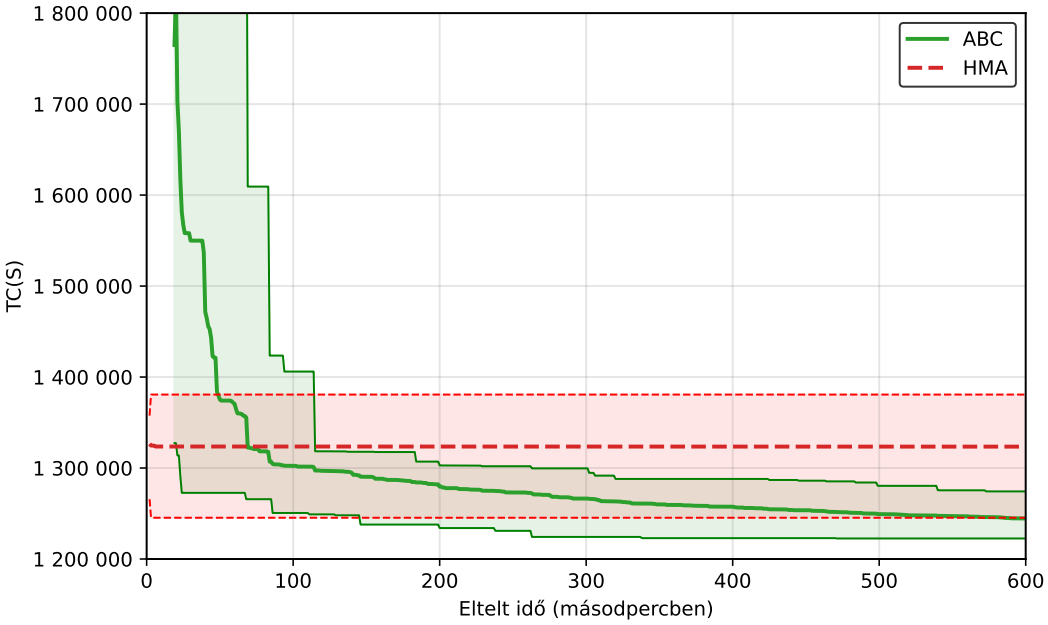
D.3. ábra. A HMA, az ACOPR és az ABC algoritmus 30 független futtatásának konvergencia sebessége az „egl-e1-A” példán, egy diagramon ábrázolva, 10 perc időkorláttal



D.4. ábra. A HMA, az ACOPR és az ABC algoritmus 30 független futtatásának konvergencia sebessége az „egl-s1-A” példán, egy diagramon ábrázolva, 10 perc időkorláttal

D.6. táblázat. A HMA és az ABC algoritmus globálisan legjobb megoldásának teljes költségének statisztikái az „egl-s1-A” példán, különböző időkorlátok között

Algoritmus	**Statisztikai mérőszám**	**Kimenetek különböző időkorlátok esetén**		
		$\leq$ 1 perc	**$\leq$ 5 perc**	**$\leq$ 10 perc**
ABC	Minimum	5 635	5 398	5 398
	Maximum	5 977	5 903	5 883
	Átlag	5 876,53	5 752,87	5 708,5
	Szórás	66,73	131,41	130,64
HMA	Minimum	5 507	5 235	5 235
	Maximum	6 628	6 611	6 401
	Átlag	6 176,57	5 738,67	5 459,87
	Szórás	355,71	440,63	209,63



D.5. ábra. A HMA és az ABC algoritmus 30 független futtatásának konvergencia sebessége az „egl-g1-A” példán, egy diagramon ábrázolva, 10 perc időkorláttal

D.7. táblázat. A HMA és az ABC algoritmus globálisan legjobb megoldásának teljes költségének statisztikái az „egl-g1-A” példán, különböző időkorlátok között

Algoritmus	**Statisztikai mérőszám**	**Kimenetek különböző időkorlátok esetén**		
		$\leq$ 1 perc	**$\leq$ 5 perc**	**$\leq$ 10 perc**
ABC	Minimum	1 272 733	1 224 289	1 222 579
	Maximum	2 069 358	1 299 609	1 274 297
	Átlag	1 370 687,2	1 266 411,97	1 244 482,33
	Szórás	161 727,50	19 121,17	13 385,36
HMA	Minimum	1 245 358	1 245 358	1 245 358
	Maximum	1 380 727	1 380 727	1 380 727
	Átlag	1 323 545,83	1 323 544,97	1 323 519,2
	Szórás	32 452,85	32 452,53	32 443,37

D.8. táblázat. A HMA és az ABC algoritmus globálisan legjobb megoldásának teljes költségének statisztikái az „egl-g2-A” példán, különböző időkorlátok között

Algoritmus	**Statisztikai mérőszám**	**Kimenetek különböző időkorlátok esetén**		
		$\leq$ 1 perc	**$\leq$ 5 perc**	**$\leq$ 10 perc**
ABC	Minimum	1 389 720	1 349 032	1 343 764
	Maximum	1 897 513	1 416 208	1 385 876
	Átlag	1 527 504,47	1 386 266,17	1 366 140,43
	Szórás	163 125,66	7 587,12	11 848,98
HMA	Minimum	1 356 204	1 356 204	1 356 204
	Maximum	1 478 279	1 478 279	1 478 279
	Átlag	1 429 353	1 425 706,43	1 424 908,2
	Szórás	30 287,61	33 358,28	34 299,91

D.7.2. A feltárásra összpontosító CARP-ABC algoritmus és az RR1 algoritmus hatékonyságának vizsgálata az esemény generátort használó DCARP keretrendszer használatával

D.9. táblázat. Az RR1, az ABC és a HMA 15 független futtatása során egy perc alatt kiszámított legjobb teljes költség, miután az „egl-e1-A" példában egy véletlenszerű **„feladat megjelenés" esemény** következett be

#	Paraméter értékek				Algoritmus kimenetei		
	ts	a	dem_a	sc_a	**RR1**	**ABC**	**HMA**
1	315	(25, 75)	58	16	3 889	3 702	3 941
2	315	(46, 45)	67	12	3 700	3 705	3 799
3	356	(43, 42)	11	11	3 720	3 645	3 883
4	379	(42, 57)	20	14	3 704	3 647	4 091
5	406	(2, 1)	62	32	3 642	3 658	3 671
6	409	(73, 74)	40	25	4 077	3 991	4 202
7	419	(9, 8)	37	26	3 693	3 709	3 709
8	427	(32, 31)	5	58	4 248	4 227	4 235
9	436	(24, 22)	64	4	3 667	3 667	3 667
10	439	(15, 14)	99	7	3 872	3 905	3 905
11	457	(6, 5)	46	8	3 714	3 714	3 714
12	490	(41, 40)	207	9	3 764	3 764	3 764
13	517	(22, 75)	66	24	3 866	3 866	3 841
14	520	(21, 51)	89	2	3 767	3 767	3 767
15	522	(39, 35)	67	7	3 622	3 622	3 622

D.10. táblázat. Az RR1, az ABC és a HMA 15 független futtatása során egy perc alatt kiszámított legjobb teljes költség, miután az „egl-e1-A” példában egy véletlenszerű **„kereslet növekedés” esemény** következett be

#	Paraméter értékek				Algoritmus kimenetei		
	ts	a	dem_a	sc_a	**RR1**	**ABC**	**HMA**
1	326	(32, 34)	36	36	4 318	3 931	4 171
2	344	(54, 52)	11	11	3 749	3 648	3 907
3	345	(50, 52)	15	15	3 753	3 636	3 678
4	374	(52, 54)	9	9	3 747	3 630	3 814
5	376	(68, 66)	32	32	4 036	3 762	3 930
6	384	(44, 45)	18	18	3 905	3 668	3 905
7	415	(46, 47)	9	9	3 975	3 590	3 590
8	431	(44, 59)	11	11	3 885	3 632	3 766
9	449	(32, 35)	12	12	4 280	3 636	3 683
10	468	(35, 32)	65	65	4 334	4 326	4 326
11	490	(44, 46)	2	2	3 636	3 550	3 550
12	490	(32, 33)	28	28	4 289	3 760	3 674
13	493	(59, 44)	5	5	3 879	3 626	3 553
14	516	(35, 32)	24	24	4 293	3 756	3 572
15	545	(35, 41)	13	13	3 775	3 657	3 559

D.11. táblázat. Az RR1, az ABC és a HMA 15 független futtatása során egy perc alatt kiszámított legjobb teljes költség, miután az „egl-e1-A” példában egy véletlenszerű **„jármű meghibásodás” esemény** következett be

#	Paraméter értékek		Algoritmus kimenetei		
	ts	h	**RR1**	**ABC**	**HMA**
1	305	2	4 124	4 217	4 737
2	311	1	4 060	4 012	4 365
3	342	2	4 204	4 126	4 585
4	344	0	4 096	4 112	4 303
5	364	0	4 096	4 112	4 241
6	399	4	4 004	4 020	4 308
7	430	1	3 966	3 966	3 966
8	451	2	4 282	4 261	4 261
9	463	0	3 718	3 652	3 639
10	490	2	4 282	4 261	4 261
11	495	2	4 282	4 261	4 261
12	506	0	3 621	3 621	3 585
13	507	1	3 874	3 736	3 572
14	523	2	4 261	4 261	4 261
15	540	2	4 282	4 282	4 282

D.7.3. A feltárásra összpontosító CARP-ABC algoritmus és az RR1 algoritmus hatékonyságának vizsgálata a forgalom szimulációt használó DCARP keretrendszer használatával

D.12. táblázat. Teljes költség forgalmi változások esetén 12 DCARP forgatókönyvön vizsgálva, ha nincs DCARP optimalizátor használva, illetve HyLS algoritmus vagy ABC algoritmus használata esetén

I	nincs	HyLS	ABC			
			Átlag	Minimum	Maximum	Szórás
1	44 029	38 806	36 303	34 418	38 130	1 503
2	38 198	32 587	37 329	30 649	50 939	7 433
3	119 952	38 088	39 329	37 380	42 399	1 934
4	52 271	32 346	34 536	33 544	36 224	901
5	47 212	37 595	36 414	33 541	40 907	2 075
6	25 064	25 266	25 266	25 266	25 266	0
7	43 572	44 822	39 862	35 772	53 643	5 159
8	34 077	40 725	33 139	29 934	39 149	3 142
9	45 510	38 656	40 927	37 068	45 171	3 010
10	54 776	33 678	33 960	31 757	41 323	2 916
11	37 585	39 970	34 498	31 993	36 427	1 779
12	24 835	24 835	24 835	24 835	24 835	0

D.13. táblázat. Szimuláció időtartama (másodpercben mérve) forgalmi változások esetén 12 DCARP forgatókönyvön vizsgálva, ha nincs DCARP optimalizátor használva, illetve HyLS algoritmus vagy ABC algoritmus használata esetén

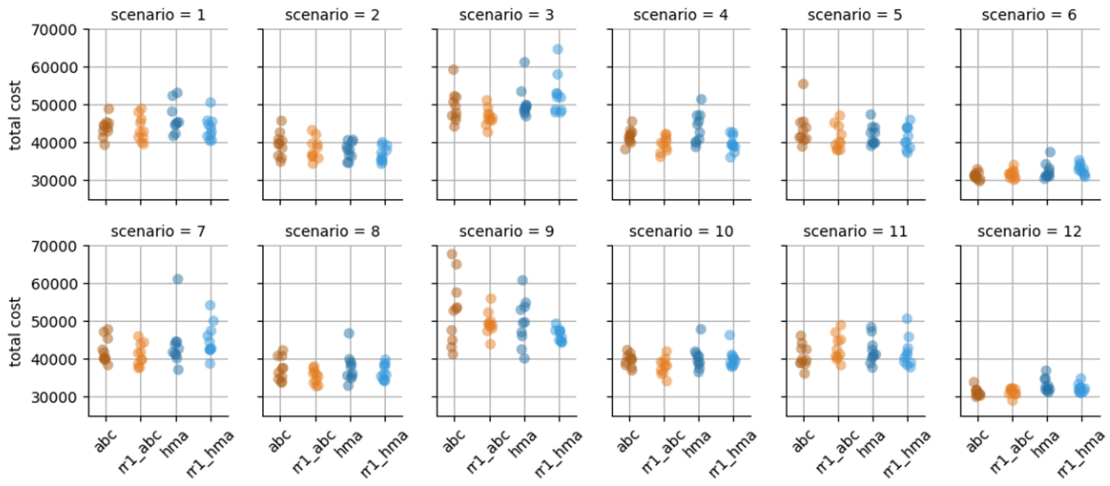
I	nincs	HyLS	ABC			
			Átlag	Minimum	Maximum	Szórás
1	4 342	3 723	3 179	2 495	4 351	654
2	4 626	5 648	7 542	3 403	15 247	4 469
3	17 261	2 482	3 012	2 505	3 861	464
4	10 522	2 868	3 827	2 779	4 945	683
5	5 214	3 880	3 897	2 886	5 828	881
6	1 832	2 012	2 012	2 012	2 012	0
7	5 245	9 573	4 644	3 072	7 795	1 561
8	4 002	7 977	4 500	3 260	8 340	1 697
9	3 408	2 962	3 268	2 518	4 018	647
10	6 733	3 115	3 723	2 712	9 040	1 918
11	4 891	7 316	3 104	2 432	4 519	627
12	1 701	1 701	1 701	1 701	1 701	0

D.14. táblázat. Teljes költség forgalmi változások és 10 „feladat megjelenés” esemény bekövetkezése esetén 12 DCARP forgatókönyvön vizsgálva, ha a használt DCARP optimalizátor az RR1, a HyLS, vagy az ABC algoritmus

I	**RR1**	**HyLS**	**ABC**			
			Átlag	**Minimum**	**Maximum**	**Szórás**
1	43 950	58 660	44 140	42 621	47 396	1 915
2	38 627	38 670	35 893	35 026	37 089	752
3	43 570	50 465	48 752	44 121	54 672	4 092
4	39 852	40 210	36 840	34 293	40 491	2 668
5	42 891	38 311	37 611	36 207	39 045	1 288
6	28 539	27 923	29 427	28 151	30 634	983
7	43 532	39 972	45 952	40 773	51 831	4 641
8	35 216	35 040	36 405	33 630	39 011	2 007
9	41 195	40 129	46 098	44 438	47 858	1 287
10	36 228	40 121	36 790	34 958	38 730	1 510
11	38 834	39 786	39 609	37 572	44 007	2 533
12	28 041	31 664	30 778	30 290	31 749	562

D.15. táblázat. Szimuláció időtartama (másodpercben mérve) forgalmi változások és 10 „feladat megjelenés” esemény bekövetkezése esetén 12 DCARP forgatókönyvön vizsgálva, ha a használt DCARP optimalizátor az RR1, a HyLS, vagy az ABC algoritmus

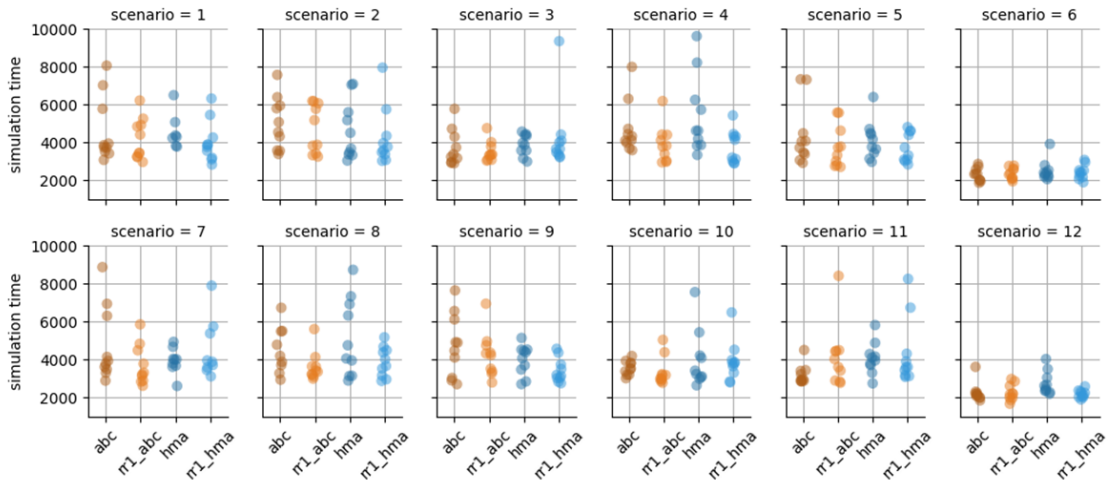
I	**RR1**	**HyLS**	**ABC**			
			Átlag	**Minimum**	**Maximum**	**Szórás**
1	3 807	6 172	5 100	3 416	9 615	2 632
2	4 785	4 301	4 390	3 353	5 165	808
3	3 157	5 085	4 357	3 084	7 733	1 927
4	3 625	4 699	3 598	2 889	4 236	522
5	3 834	4 045	3 057	2 695	3 832	462
6	2 284	1 975	2 188	1 938	2 611	280
7	3 352	3 419	4 989	3 575	7 802	1 679
8	4 372	3 660	4 716	3 923	5 561	694
9	3 239	3 054	4 022	3 008	6 552	1 457
10	3 308	5 510	3 600	3 111	4 053	391
11	3 685	3 405	3 783	3 087	6 035	1 271
12	2 249	3 262	2 645	2 111	3 333	483



D.6. ábra. Teljes költség 10 független futás végén, forgalmi változások és 10 „feladat megjelenés" esemény bekövetkezése esetén 12 DCARP forgatókönyvön vizsgálva, ha a használt DCARP optimalizátor az ABC, az RR1 & ABC, a HMA, vagy az RR1 & HMA algoritmus kombinációk

D.16. táblázat. Átlagos teljes költség forgalmi változások és 10 „feladat megjelenés" esemény bekövetkezése esetén 12 DCARP forgatókönyvön vizsgálva, ha a használt DCARP optimalizátor az ABC, az RR1 & ABC, a HMA, vagy az RR1 & HMA algoritmus kombinációk

I	**ABC**	**RR1 & ABC**	**HMA**	**RR1 & HMA**
1	43 838	43 565	51 183	43 757
2	39 418	38 244	37 846	36 965
3	49 526	46 659	50 408	52 519
4	41 551	39 474	43 800	39 643
5	43 626	41 181	41 892	41 259
6	31 062	31 574	32 322	32 824
7	42 272	41 068	43 773	45 083
8	37 228	35 286	37 671	36 462
9	52 658	49 329	49 706	46 287
10	39 620	37 956	40 306	40 052
11	40 718	42 859	42 086	41 638
12	31 084	31 122	32 980	32 117



D.7. ábra. Szimuláció időtartama (másodpercben mérve) 10 független futás végén, forgalmi változások és 10 „feladat megjelenés” esemény bekövetkezése esetén 12 DCARP forgatókönyvön vizsgálva, ha a használt DCARP optimalizátor az ABC, az RR1 & ABC, a HMA, vagy az RR1 & HMA algoritmus kombinációk

D.17. táblázat. Szimuláció átlagos időtartama (másodpercben mérve) forgalmi változások és 10 „feladat megjelenés” esemény bekövetkezése esetén 12 DCARP forgatókönyvön vizsgálva, ha a használt DCARP optimalizátor az ABC, az RR1 & ABC, a HMA, vagy az RR1 & HMA algoritmus kombinációk

I	**ABC**	**RR1 & ABC**	**HMA**	**RR1 & HMA**
1	4 899	4 194	7 879	4 019
2	4 788	4 703	5 294	4 228
3	3 709	3 507	3 531	4 247
4	5 386	3 903	4 601	3 775
5	4 257	3 784	4 145	3 745
6	2 401	2 352	2 611	2 419
7	5 805	3 693	6 821	5 578
8	4 218	3 627	5 497	3 889
9	6 025	4 249	4 209	3 410
10	3 572	3 353	3 673	3 900
11	3 390	4 132	4 120	4 321
12	2 336	2 249	2 643	2 180

D.18. táblázat. Teljes költség forgalmi változások és 20 „feladat megjelenés” esemény bekövetkezése esetén 12 DCARP forgatókönyvön vizsgálva, ha a használt DCARP optimalizátor az RR1, az ABC, vagy a HMA algoritmus

I	RR1	ABC				HMA			
		Átlag	**Min.**	**Max.**	**Szórás**	**Átlag**	**Min.**	**Max.**	**Szórás**
1	51 592	43 691	39 329	45 277	2 470	55 190	41 611	94 473	22 366
2	38 835	38 179	34 842	40 769	2 558	38 642	36 289	40 628	1 755
3	49 817	49 350	44 173	59 245	5 850	48 566	46 873	49 802	1 112
4	39 662	42 453	40 281	45 522	2 108	44 265	39 794	47 137	3 137
5	54 099	43 587	38 854	55 445	6 710	42 744	39 722	47 335	3 076
6	31 837	31 239	30 103	32 854	1 115	32 993	30 946	37 396	2 620
7	43 829	44 457	39 626	47 789	3 413	46 558	40 993	61 083	8 275
8	34 686	36 784	33 612	40 743	2 806	39 209	35 500	46 727	4 539
9	55 367	57 258	47 548	67 723	8 666	51 447	46 019	54 854	3 603
10	39 444	40 036	38 331	42 278	1 522	39 286	36 455	41 374	1 917
11	42 620	41 882	38 681	46 095	2 919	42 637	37 648	48 361	4 894
12	31 544	31 161	30 216	33 776	1 473	32 051	31 091	32 823	689

D.19. táblázat. Szimuláció időtartama (másodpercben mérve) forgalmi változások és 20 „feladat megjelenés” esemény bekövetkezése esetén 12 DCARP forgatókönyvön vizsgálva, ha a használt DCARP optimalizátor az RR1, az ABC, vagy a HMA algoritmus

I	RR1	ABC				HMA			
		Átlag	**Min.**	**Max.**	**Szórás**	**Átlag**	**Min.**	**Max.**	**Szórás**
1	8 834	4 899	3 068	8 065	2 037	7 879	3 817	15 836	5 388
2	4 978	4 788	3 551	5 946	1 169	5 294	3 477	7 096	1 752
3	3 469	3 709	2 890	5 774	1 203	3 531	2 984	4 385	543
4	3 337	5 386	3 585	7 998	1 767	4 601	3 333	6 255	1 063
5	5 149	4 257	3 056	7 341	1 761	4 145	2 963	6 401	1 335
6	2 333	2 401	1 856	2 854	442	2 611	2 047	3 914	753
7	4 674	5 805	3 297	8 875	2 349	6 821	3 611	17 555	6 020
8	3 584	4 218	2 933	5 487	980	5 497	3 127	8 734	2 296
9	6 564	6 025	4 892	7 646	1 167	4 209	3 470	4 523	455
10	3 728	3 572	3 011	4 183	443	3 673	2 635	5 434	1 147
11	3 522	3 390	2 853	4 501	662	4 120	2 741	5 819	1 108
12	2 506	2 336	1 842	3 610	722	2 643	2 266	3 499	501