Doktori értekezés

Periodikus ipari folyamatok erőforrás eloszlását optimalizáló algoritmusok és hatékony szoftverimplementációik kutatása

DOI:10.18136/PE.2020.733

Szerző:
Bartos Anikó

Témavezető:
Dr. Bertók Botond

Pannon Egyetem
Műszaki Informatikai Kar
Informatikai Tudományok Doktori Iskola

2019

Periodikus ipari folyamatok erőforrás eloszlását optimalizáló algoritmusok és hatékony szoftverimplementációik kutatása

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

Írta: Bartos Anikó

Témavezető: Dr. Bertók Ákos Botond

Elfogadásra javaslom (igen / nem)

……………………….
Dr. Bertók Ákos Botod
(témavezető)

A jelölt a doktori szigorlaton .... %-ot ért el,
Veszprém, 2018. 09. 20.

……………………….
(a Szigorlati Bizottság elnöke)

Az értekezést bírálóként elfogadásra javaslom:

Bíráló neve: …........................ …................. igen /nem

……………………….
(aláírás)

Bíráló neve: …........................ …..................) igen /nem

……………………….
(aláírás)

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

Veszprém,

……………………….
(a Bíráló Bizottság elnöke)

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

……………………….
(az EDHT elnöke)

Tartalomjegyzék

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani mindazoknak, akik lehetővé tették, hogy ez a dolgozat elkészüljön. Elsősorban témavezetőmnek, Dr. Bertók Botondnak tartozom köszönettel, aki éveken át irányította munkámat, és értékes tanácsokkal látott el. Dr. Hegyháti Máténak szakmai támogatásáért és a rengeteg segítségért vagyok hálás. Riz Barnabásnak nem csak az idegennyelvi lektorálást szeretném megköszönni, de a dolgozat elkészülése közbeni személyes támogatását is. Továbbá szeretnék köszönetet mondani kollégáimnak a Rendszer- és Számítástudományi Tanszéken, akik munkám elvégzéséhez hozzájárultak, valamint családomnak és barátaimnak a biztatásért.

Kivonat

Napjainkban a globális verseny miatt a piaci szereplőknél alapvető fontosságú, hogy az ipari és üzleti folyamatok hatékonyak legyenek. Egy vállalat csak akkor maradhat hosszú távon is életképes, ha folyamatai optimalizálva vannak, és a rendelkezésére álló erőforrásokból a lehető legtöbbet tudja kihozni. Ezek a folyamatok azonban túlságosan is összetettek ahhoz, hogy számítógépes támogatás nélkül átláthatóak legyenek, még kisvállalatok esetében is. A számítógéppel történő döntéstámogatás nagyban megkönnyítheti a vállalati vezetők és irányítók munkáját, és, ahogy egy dolgozatban megtalálható példa mutatja, akár 25%-os hatékonyságjavulást is eredményezhet. A P-gráf módszertan egy olyan modellező és optimalizáló eszközt biztosít, amely az elmúlt évtizedekben már számos területen bizonyította hatékonyságát, és egy probléma legjobb, illetve N-legjobb megoldását visszaadva gyakran szolgáltatott alapot döntéstámogató rendszerekhez is.

A legtöbb vállalatnál azonban az üzleti tervezés nem egyszeri, hanem az újabb információk alapján rendszeresen megismételt. Ezek az időszakok ugyanakkor nem függetlenek egymástól, így szükséges a különböző periódusokra különböző pontosságú, gördülő tervezést használni. Új eredményként multiperiodikus tervezést támogató, optimalizáló modellek generálására és megoldására dolgoztam ki módszereket, melyeket a dolgozatban bemutatok. Az eredmények között egyaránt megtalálhatóak általános célú, illetve különböző feladatokra specializált metódusok, mint például a gyártástervezés, gyártósor kiegyensúlyozás, valamint az energiaellátás elosztás.

A hosszútávú tervezés jelentősen csökkentheti a felmerülő kockázatokat, ellenben nagy mértékben növeli a megoldandó optimalizálási feladat méretét. Az ilyen, nagy méretű feladatok megoldásához elengedhetetlen egy olyan, hatékony megoldó algoritmus használata, mely maximálisan kihasználja a döntéstámogató informatikai rendszer számítási kapacitását. Munkám eredményeként dolgozatomban egy általános célú, a fenti feladatok mindegyikének megoldására képes folyamatoptimalizáló algoritmus párhuzamos megvalósítását vezetem be, mely peer-to-peer típusú együttműködő feladatelosztást és automatikus adaptív paraméterhangolást is tartalmaz.

Abstract

Nowadays, due to the global competition, it is crucial for market participants that its industrial and business processes should be effective. A company can only remain viable in the long term if its processes are optimized, and get the most out of the resources at its disposal. However, these processes are too complex to be transparent without computer support, even for small businesses. Computerized decision support can greatly facilitate the work of corporate executives and managers, and as in the example in this dissertation, can lead to efficiencies of up to 25%. The P-Graph methodology provides a modelling and optimization tool that has proven its effectiveness in various areas over the past decades and has often provided a basis for decision support systems by returning the best or N-best solution to a problem.

In most companies, however, business planning is not a one-off but repeated on the basis of new information. However, these periods are not independent of each other, so it is necessary to use different precision rolling plans for different periods. As a new result, I have developed methods for generating and solving optimization models supporting multi-period planning, which is presented in this thesis. The results include both general-purpose as well as specialized tasks, such as production planning, line balancing, or energy distribution.

Although long-term planning can significantly reduce the risks involved, it greatly increases the size of the optimization problem to be solved. In order to solve such large-scale tasks it is necessary to use an effective solving algorithm that takes full advantage of the computational capacity of the decision support system. As a result of my work, I introduce a parallel implementation of a process optimization algorithm capable of solving each of the above tasks, which includes peer-to-peer collaborative task allocation and adaptive parameter setting.

Abstracto

Hoy en día, debido a la competencia global es fundamental para los partícipes del mercado que sus procesos industriales y de negocios sean efectivos. Una empresa solo puede seguir siendo viable a largo plazo si sus procesos están optimizados, y aprovechar al máximo los recursos a su disposición. Sin embargo, estos procesos son demasiado complejos para ser transparentes sin soporte informático, incluso para pequeñas empresas. El soporte de decisiones por ordenador puede facilitar enormemente el trabajo de los ejecutivos y gerentes corporativos, y como en el ejemplo de esta disertación, puede llevar a eficiencias de hasta el 25 %. La metodología P-Graph proporciona una herramienta de modelado y optimización que ha demostrado su efectividad en varios ámbitos durante las últimas décadas y a menudo ha proporcionado una base para los sistemas de soporte de decisiones al devolver la mejor o la N-mejor solución a un problema.

No obstante, en la mayoría de las empresas, la planificación comercial no es un hecho aislado, sino que se repite acorde a la nueva información. Pero, estos periodos no son independientes entre sí, por lo que es necesario usar diferentes planes de precisión para diferentes periodos. Como nuevo resultado, he desarrollado métodos para generar y resolver modelos de optimización que apoyan la planificación de múltiples peroodos, que se presenta en esta tesis. Los resultados incluyen tareas especializadas y de propósito general, como la planificación de la producción, el balanceo de líneas o la distribución de energía.

Aunque la planificación a largo plazo puede reducir en garn medida los riesgos involucrados, aumenta enormemente el tamaño del problema de optimización a resolver. Para resolver tales tareas a gran escala, es necesario utilizar un algoritmo de resolución eficaz, que aprovecha al máximo la capacidad computacional del sistema de apoyo a la decisión. Como resultado de mi trabajo, presento una implementación paralela de un algoritmo de optimización de procesos capaz de resolver cada una de las tareas anteriores, que incluye la asignación de tareas colaborativas punto a punto y la configuración de parámetros adaptativa.

1. fejezet

Bevezetés

A XX. század második felétől a számítógépek robbanásszerűen terjedtek el, nem csak a kutatóközpontokban és az iparban, hanem a háztartásokban is. A folyamatosan növekvő számítási kapacitású gépek segítségével számos olyan matematikai problémát sikerült megoldani, vagy igazolni, mely korábban lehetetlennek bizonyult. Számítógépek és gépesítés bevezetésével az ipar is termelékenyebb lett, és ezzel együtt egyre több helyen kezdtek el figyelni az optimalizálásra. Bár az optimum megtalálására korábban is voltak módszerek, de igazán csak a számítógépes támogatás elterjedésével kaptak nagyobb teret. Míg korábban, manuálisan nehézségekbe ütköztek egyes feladatok felírásai és megoldásai, úgy a számítógépekkel egyre inkább elterjedő modellezőknek és megoldóknak köszönhetően a legtöbb területen elérhetővé vált az optimalizálás megvalósítása.

Optimalizálást alkalmazni - főleg, ha annak eredménye kimagasló- nem csak kutatási szempontból motiváló, hanem pozitív anyagi, gazdasági és környezeti hatása lehet. A kutatási témám megválasztásánál és egyben a disszertáció elkészítésében is az inspirált, hogy ezen a területen nem csak elméleti eredményeket lehet elérni, de a megfelelő algoritmusok és módszerek a gyakorlatban is hasznosíthatók és erre igény is van. A következő néhány fejezetben bemutatásra kerül a folyamathálózat-szintézis, mint optimalizálási módszertan, valamint ehhez a területhez kapcsolódó kutatásaim és eredményeim. Többek között ilyenek a több periódusból álló modellek, a gyártásoptimalizálás vagy a megoldó algoritmusok fejlesztése.

2. fejezet

Feladatmegfogalmazás

A képzésben eltöltött éveim alatt lehetőségem adódott számos projektmunkában részt venni. Munkám során az ipari együttműködések keretében - például a Foxconn-nal, vagy SAGEM-el - jó néhány valós életbeli döntéstámogatási feladattal találkozhattam, mely feladatok, akár a kisebb vállalatoknál felmerülők is, szinte mindegyike jellegéből és összetettségéből adódóan valamilyen számítógépes támogatást igényelt. A felmerülő kérdésekre a válaszokat keresve azt találtam, hogy az irodalomban megtalálható modellekhez és megoldásokhoz sokszor kell valamilyen kiegészítés, vagy akár egy újfajta megközelítés, modell felírása. Úgy vélem, hogy a megtalálható megoldásokhoz aktívan hozzá tudok járulni a munkám során elért és itt bemutatott új eredményeimmel. A dolgozatomban ezeket a feladatokat, és a feladatokra adott válaszokat, megoldásokat fogom bemutatni.

3. fejezet

Szakirodalmi áttekintés

Ebben a fejezetben bemutatásra kerülnek azok a korábban már publikált módszerek és megoldások, melyek alapjául szolgáltak a kutatási tevékenységemnek, és amelyekre építve hoztam létre az új eredményeimet. Ismertetésre kerülnek mind a metódusok előnyei, mind pedig hátrányai, vagy esetleg hiányosságai, amikre adott válaszaim a dolgozat lényegi részét képezik. A fejezet során ismertetni fogom az ütemezési feladatokat, azon belül is az ütemezési feladatok általános megfogalmazását, a MILP és az állapottér bejáráson alapuló megoldásokat, valamint az S-gráf módszertant. A fejezet áttekintést ad a folyamathálózat-szintézisről (Process Network Synthesis, PNS), annak alapjairól, valamint az alkalmazási területekről, ahol hatékonyan alkalmazták már, továbbá a modellezésére és megoldására szolgáló szoftverről és annak algoritmusairól, és azok korábbi párhuzamosításáról.

3.1. Ütemezési feladatok

Ütemezési feladatok széles körben fordulnak elő az élet számos területén, legyen szó mezőgazdaságról, gyártásról, fuvarozásról vagy akár informatikai rendszerekről. Fontos, hogy rendelkezésre álljanak olyan módszerek, algoritmusok és szoftverek, amelyek segítségével optimális vagy közel optimális megoldást lehet kapni ezekre a problémákra. Mivel ütemezési feladatok esetén az egyes lehetséges megoldások minősége nagy mértékben eltérhet, különösképpen fontos, hogy a lehető legjobb megoldást találjuk meg a problémára. Fontos szempont az is, hogy egy ilyen megoldó a felhasználó számára elfogadható időn belül legyen képes eredményt adni.

Az ütemezési feladatok célja többféle is lehet. Az egyik megközelítés az, hogy a lehető legrövidebb idő alatt történjen meg az előre meghatározott, adott feladatok elvégzése, szakaszos működésű berendezésekkel (például számítógépgyárban 100 gép összeszerelése), de lehet olyan megközelítés is, ahol adott (fix) idő alatt kell a lehető legnagyobb profitot elérni (például egy műszak alatt összeszerelendő számítógépek maximalizálása). Az ütemezési problémát tovább bonyolítja, ha egy adott lépést, feladatot több műveleti egység is képes megoldani, vagy ha esetleg vannak átállási- és határidők. A feladatoknál felmerülhetnek különböző költségek, és az sem mindegy, hogy milyen tárolási stratégiát lehet alkalmazni, ebből kifolyólag az ütemezési feladatok megoldása nem triviális.

Az ilyen feladatokat kezdetben heurisztikus szabályok alapján próbálták megoldani, amely módszerek fő hátránya, hogy nem tudják garantálni az optimális megoldást. Manapság már képesek vagyunk egy-egy ütemezési feladatosztályhoz egy speciális matematikai modellt felírni, viszont egy általános, minden feladatosztályhoz jó megoldó nem áll rendelkezésre. Ez azért van, mert a gyakorlatban egy ilyen feladatnál rengeteg változót kell figyelembe venni és bevezetni a modell felírásánál, ráadásul a optimalizálás céljai is nagy mértékben eltérhetnek. A legáltalánosabban a vegyes egészértékű lineáris optimalizálási modellek használhatóak, amelyekre már több, általános megoldó szoftver is létezik. A fejezet további részében bemutatásra kerülnek az ütemezési feladatok legelterjedtebb fajtái, majd ismertetem a kapcsolódást a folyamathálózat-szintézissel, bemutatom a PNS alapjait, alkalmazási területeit és az eddig publikálásra került eredményeit.

3.1.1. Ütemezési feladatok általános megfogalmazásai

Az ütemezési feladatok megértéséhez az alábbi alapfogalmakat kell bevezetni:

– **Recept**: A recept azokat a minimális információkat tartalmazza, amik a termékek előállításához szükségesek. Ilyen például a feladatok hálózata és az azokhoz rendelhető berendezések is, ami tartalmazza továbbá azt, hogy egy berendezés egy feladatot mennyi idő alatt végez el. A receptben egy feladathoz több bemenet és kimenet is tartozhat.

  - **Termék** vagy **cél**: Terméknek vagy célnak nevezzük azokat az anyagokat, tárgyakat, esetleg állapotokat, amiket a gyártórendszerben elő kívánunk állítani, vagy el akarunk érni.
  - **Berendezés**: Olyan eszköz vagy erőforrás, amit a termékek előállításához fel lehet használni, és rendelkezésre állását tekintve megújuló, vagyis használata után újra rendelkezésre áll.
  - **Feladat**: A feladatok olyan elemi tevékenységek, melyek további részekre már nem bonthatók. Meghatározott idő alatt állítják elő az adott inputból az adott outputot. Egy feladat végrehajtásához egy vagy több berendezés is rendelkezésre állhat, és a végrehajtási ideje függ attól, hogy melyik berendezéssel kerül megvalósításra. Egy berendezés egy időben csak egy feladatot végezhet el, és egy feladathoz csak egy berendezés rendelhető.

– **Adag**: Ha az előállítandó termék mennyisége több, mint amennyi a recept egyszeri végrehajtásából keletkezik, akkor a recept többszöri alkalmazásával lehet előállítani az igényelt termékmennyiséget. A recept egyszeri végrehajtásával egy adagnyi termék keletkezik.

A gyártás során keletkező félkész termékeknél jogosan merül fel a kérdés, hogy ezeket lehet-e valahol tárolni a gyártás folyamata közben, akár valamilyen tárolóban, raktárban, akár magában a műveletet végző berendezésben (például gyártószalag). A tárolásnál fontos, hogy a tárolási kapacitás korlátozott-e, esetleg a tárolási időre vonatkozóan vannak-e megkötések. Ezek mentén két csoportba lehet sorolni a tárolási stratégiákat.

– Stratégiák tárolási idő szempontjából

- **UW** (Unlimited wait) stratégia: Ez az egyik legegyszerűbb és legáltalánosabb stratégia. Ekkor a tárolandó anyag az idővel nem veszít a tulajdonságaiból, így végtelen ideig tárolható, mielőtt a következő feladatra lépne.
- **LW** (Limited wait) stratégia: Itt meg van határozva egy bizonyos idő, aminél tovább a köztes anyagok nem tárolhatóak anélkül, hogy vesztenének tulajdonságaikból.
- **ZW** (Zero wait) stratégia: Ez tekinthető az LW stratégia egy speciális esetének is, ahol a tárolási idő 0. Vagyis semmilyen mód nincs a tárolásra, és a legutolsó módosítást végző berendezésben sem történhet tárolás, várakozás. Ekkor a félkész terméket azonnal a következő folyamatra kell küldeni.

– Stratégiák tárolási kapacitás szempontjából

- **UIS** (Unlimited intermediate storage) stratégia: A félkész termékeket ebben az esetben -gyakorlati szempontból tekintve- végtelen mennyiségben lehet tárolni. (Ilyen lehet egy bérelt raktárterület, ahol telítettségkor további terület bérelhető.)
- **NIS** (Non intermediate storage) stratégia: Ebben az esetben a tárolás, várakozás a gyártóberendezésben történik, mivel nincs erre dedikált külön hely. Ekkor a berendezést, amiben tárolás történik csak akkor lehet újra gyártásra használni, ha a benne lévő félkész terméket eltávolítottuk.
- **FIS** (Finite intermediate storage) stratégia: Ebben az esetben is van köztes tárolásra lehetőség, de a tárolók száma és mérete véges.

A valóságban elképzelhető ezen tárolási stratégiák valamilyen szintű kombinációja akár különkülön is az egyes félkész termékekre. A tárolók a több periódusú modelleknél is kulcsszerepet játszanak, melyek későbbi fejezetekben kerülnek részletezésre.

3.1.2. MILP megoldások

A lineáris és nemlineáris programozási feladatokat tekintve a vegyes egészértékű, vagyis MILP (Mixed Integer Linear Programming), illetve a MINLP (Mixed Integer Non-linear Programming) modellek a legintenzívebben kutatott és publikált módszerek között vannak, és meghatározó szerepet töltenek be az ütemezési feladatoknál is. Ebben a fejezetben bemutatásra kerülnek ezeknek a problémáknak a legismertebb típusai.

Idő-diszkretizáción alapuló ütemezés

Az idő-diszkretizáción alapuló megoldások az elsők között kerültek publikálásra [1]. Ezek alapja, hogy a teljes időintervallumon időpontok vagy időszeletek kerülnek definiálásra (Például egy nap ütemezése órás bontásban.). A két módszer nagyon hasonló és az átjárás közöttük könnyen megoldható, hiszen két időpont közötti részt lehet időszeletnek tekinteni, és az időszeletek kezdési ideje egy időpontot jelöl. Az időszeletes felbontás általában szekvenciális folyamatokra, míg az időpontokra bontás általánosabb hálózati problémákra jó. Ezeknél a problémáknál az időpontokhoz bináris változók tartoznak, amelyek azt reprezentálják, hogy adott műveleti egységek az abban az időpontban elkezdenek-e végrehajtani egy műveletet.

Mivel jól látható, hogy ezek a bináris változók az időpontok és feladatok számával ugrásszerűen növekednek, fontos, hogy a probléma leírásra olyan modell legyen megadva, ahol ezen időpontok száma a lehető legkisebb, de mégis elegendőek a megfelelő optimalizáláshoz. Az egyik ilyen módszer, ha kezdetben csak kevés időpont szám mellett történik az optimalizálás, majd az időpontok számát egyesével növelve újra és újra megvalósul az optimalizálás mindaddig, amíg két, egymást követő iteráció után ugyanaz az eredmény nem születik. Kellően sok időponttal már megközelítőleg jól fel lehet írni egy modellt, de a túl sok időpont lassítja is ezt a folyamatot. A modellalkotók ebből adódóan megpróbálnak különböző technikák alkalmazásával kevesebb időpont használatával olyan modelleket adni, ami gyorsabban, de még mindig megtalálja a megoldást. Ezekben az gyorsításokban viszont megvan az esély arra, hogy véletlenül pont az optimumot fogják kizárni. Az idő-diszkretizáción alapuló modellek egyik előnye, hogy széles körben használhatók, hiszen a precedencia alapú modellekkel szemben nem kell a megoldás előtt, előre ismerni a feladatok, termékek számát. A módszer továbbá megengedi, hogy ugyanazt a feladatot több gép is végezhesse párhuzamosan.

Sorrendiségen alapuló modellek

A sorrendiségen, vagy precedencián alapuló modellek esetében nem szükséges, hogy az időhorizont felosztva, diszkretizálva legyen, aminek köszönhetően a modellben nem lesz ismeretlen paraméter. Előnye, hogy amely problémát képesek megoldani, azt sokkal nagyobb hatékonysággal teszik. Itt kulcsszerephez jut két, bináris változókat tároló halmaz: $Y_{i,j}$ valamint $X_{i,j,i'}$. Előbbi azt mondja meg, hogy egy i-edik feladat hozzá van-e rendelve j-edik műveleti egységhez, míg utóbbi értékei akkor lesznek '1'-ek, ha mind i mind pedig i' a j-ben van elvégezve, és i hamarabb befejeződik, mint az i' szekvencia.

A sorrendiségen alapuló modelleknek két fő változata van: azonnali és általános modellek. Az általános modellnél a bináris változók felére van csak szükség ($X_{i,j,i'}$ komplementere az $X_{i',j,i}$), az azonnali előnye viszont, hogy abban számos jellemzőt egyszerűbb kifejezni. Léteznek hibrid modellek is, amik ezeket a módszereket ötvözik.

3.1.3. Állapottér bejáráson alapuló megoldások

Diszkrét eseményű rendszerek modellezésére széles körben használt módszerek az automaták és Petri-hálók[2]. A modellezésen túl kísérletek voltak arra, hogy ezekben a módszerekben rejlő lehetőségeket kiaknázva az ütemezési problémákat meg is oldják velük. Ilyen az időzítéses Petri-háló [3], ami már több ütemezési problémához is elegendő. Az ilyen megoldók legtöbbször Branch-and-Bound algoritmust használnak az optimális megoldás megtalálásához.

Modellezéskor fontos, hogy az esetleges hibákat elkerüljük, és ne juthasson holtpontra a feladat. Az állapottér bejáráson alapuló modellek előnye, hogy maga a modellezés elég egyértelmű. Ennek a tulajdonságának köszönhetően jól hasznosíthatók a vezérlési döntési rétegben, hiszen könnyű az integrációjuk, valamint, jellegükből fakadóan újraütemezésekhez is használhatóak, azonban hatékonyságukban még mindig elmaradnak a MILP vagy S-gráf algoritmusok mögött.

3.1.4. S-gráf módszertan

Az S-gráf módszertant elsőként 1998-ban használták ütemezési feladatra [4]. A modell ebben az esetben egy irányított gráfból, az S-gráfból áll, és ez a gráf reprezentálja mind a recepteket, mind pedig a részleges vagy teljes ütemezést [5]. A gráfban a termékek és a feladatok csúcspontokként kerülnek megjelenítésre. Ha két művelet között függőség, egymásra épülés van, akkor az őket ábrázoló csúcsok között is lesz irányított él, és ezek az élek reprezentálják a köztük lévő függőségeket is.

Az összes S-gráf algoritmus kiterjeszti a gráfot úgynevezett ütemezési élekkel, amik az algoritmusok által generált ütemezési döntéseket reprezentálják. Minden egyes csúcspontnál igaz az, hogy ha már megtörtént a döntés, akkor a csúcspont halmaza le lesz cserélve a kiválasztott egységgel. A gráfban az ütemezési élek súlya alapesetben '0' ha nincs átállási, átviteli vagy tisztítási idő a feladatban. Ezek az ütemezési élek továbbá kifejezik, hogy nem elegendő a műveleti egységnek végeznie az aktuális anyaggal mielőtt a következőre lépne, hanem recept szerint a következő feladatnál, ahol az aktuális anyag input lesz (amit egy másik műveleti egység fog végezni) szintén szabadnak kell lennie a műveleti egységnek, hogy fogadhassa az új anyagot. Ütemezési élek nélkül az S-gráf megfelel a receptgráfnak.

3.2. A folyamathálózat-szintézis alapjai

Az iparban a gyártási és termelési folyamatoknál egy fontos kérdés, hogy hogyan kell úgy összeállítani a gyártórendszert a rendelkezésre álló és esetleg még nem álló erőforrásokból, továbbá hogyan kell ezt a rendszert konfigurálni ahhoz, hogy az előállítandó termék gyártása optimális legyen. 1992-ben Friedler és Fan professzor publikálta a folyamathálózat-szintézist, amely egy strukturális tulajdonságokra építő technika, és választ ad a vegyipari termelési folyamatok modellezésének kérdéseire és optimalizálására [6]. A megfelelő szintézis az energiafogyasztást akár 50, a költségeket pedig 35%-al is képes csökkenteni, így elmondható, hogy egy igen hatékony optimalizálási eljárás [7].

3.2.1. P-gráf módszertan alapjai, alapfogalmak

A folyamathálózat-szintézis fő építőelemei az anyagok és a műveleti egységek. A műveleti egységek anyagból vagy anyagokból valamilyen hozzáadott értékkel rendelkező új anyagot vagy anyagokat képesek előállítani, és ezen lépések sorozatával a nyersanyagokból terméket előállítani.

Jelölje M az anyagok halmazát, amely halmaz véges és nem üres. A kapcsolódó szintézis feladat egy (P, R, O) hármassal adható meg, ahol P a termékeket, R a nyersanyagokat reprezentálja, és P halmaz elemei azok az anyagok, termékek, amiket a folyamat során elő kell állítani, R elemei pedig azok az anyagok, amik a gyártás megkezdése előtt rendelkezésre állnak, és nem állítja elő őket a folyamat során használt berendezés.

$$P \subset M,\ R \subset M,\ valamint\, P \cap R = \emptyset. \tag{3.1}$$

O reprezentálja a műveleti egységeket, amik az anyagok közötti átalakításokért felelnek. Mivel O-nak mind kimenete, mind pedig bemenete is lehet, ezért egy rendezett párral, $(\alpha, \beta) \in O$ lehet modellezni ezt a kapcsolatot, ahol α jelöli a be, β pedig a kimeneti anyagok halmazát. Köztes anyagok azok az anyagok, amik be- és kimenetei is műveleti egységeknek. Ezek kiinduláskor még nem állnak rendelkezésre, viszont a gyártás során keletkeznek és felhasználásra kerülhetnek a termék előállításának érdekében. Azokat az anyagokat, amik gyártás során keletkeznek, de nem használja fel őket egy műveleti egység sem és P halmaznak sem elemei, melléktermékeknek hívják.

3.2.2. Strukturális reprezentáció, P-gráf

A kombinatorikus tulajdonságok kihasználásához a folyamat egyértelmű, strukturális reprezentációja szükséges. Az anyagok és műveleti egységek kettőssége, és ezen elemek váltakozása miatt a PNS strukturális reprezentációja egy irányított páros gráf, a P-gráf. A csúcsok két típusát az anyagok és műveleti egységek adják, és a páros gráfok tulajdonságainak köszönhetően két azonos típusú csúcs nem kapcsolódhat közvetlenül egymáshoz. A P-gráf tehát egy olyan (M,O) pár, ahol a gráf csúcsai

$$V = M \cup O. \tag{3.2}$$

A gráf élei az

$$E = E_1 \cup E_2 \tag{3.3}$$

halmaz elemei, ahol

$$E_1 := \{(x, y) | y = (\alpha, \beta) \in O,\ valamint\, x \in \alpha\} \tag{3.4}$$

és

$$E_2 := \{(y, x) | y = (\alpha, \beta) \in O,\ valamint\, x \in \beta\}. \tag{3.5}$$

A fenti, formális definícióban x az anyag típusú, y pedig a műveleti egység típusú csúcsokat jelöli. α csúcsból mutat irányított él műveleti egység típusba, β csúcsba pedig műveleti egységből mutat irányított él.

(M_1, O_1) gráf (M_2, O_2) részgráfja, azaz

$$(M_1, O_1) \subseteq (M_2, O_2) \tag{3.6}$$

ha

$$M_1 \subseteq M_2 \; valamint \; O_1 \subseteq O_2. \tag{3.7}$$

A fentiek alapján egy folyamat struktúráját meg lehet adni a P-gráf formátumnak megfelelően az M, mint anyag, és O, mint műveleti egység halmazok felhasználásával. Az anyagokat a gráf anyag típusú, míg a műveleti egységeket a műveleti egység típusú csúcsok reprezentálják. Előbbieket körökkel, utóbbiakat téglalapokkal jelölik, a 3.1. ábrának megfelelő módon. Megkülönböztetésre kerülnek a nyersanyagok, amiket háromszöggel ellátott körök, a köztes anyagok, amiket teli körök, illetve a termékek, amiket koncentrikus körök reprezentálnak.

3.1. ábra. A folyamathálózat-szintézis szimbólumai

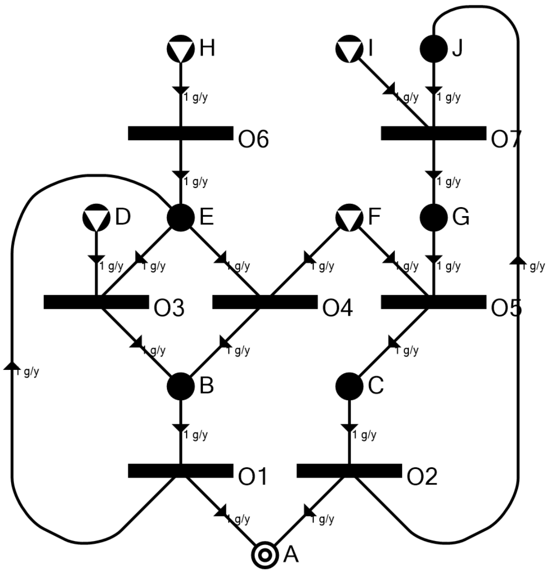
Egy műveleti egység és egy anyag típusú csúcs között akkor és csak akkor megy él, ha ez a kapcsolat része a modellezendő folyamatnak. A műveleti egységek által felhasznált és előállított anyagok arányai az élek súlyain jelennek meg. Az élek irányai a folyamat előrehaladásának irányával megegyezőek. Egy általános P-gráf reprezentáció, amely négy nyersanyagot, egy terméket, és hét műveleti egységet tartalmaz a 3.2. ábrán látható.

3.2.3. Axiómák

Minden P-gráf struktúrának teljesítenie kell az alábbi öt axióma mindegyikét:

- (S1): Minden terméknek és célnak szerepelnie kell a gráfban.
- (S2): Egy M típusú csúcs nyersanyag, ha semmilyen műveleti egység nem állítja azt elő, vagyis nincs bemenete.
- (S3): Minden O típusú csúcs, vagyis műveleti egység amelyet tartalmaz a gráf, előzetesen definiálva kell, hogy legyen a szintézis feladatban.
- (S4): Minden O típusú csúcsból, azaz műveleti egységből kell, hogy vezessen legalább egy út egy termékig.
- (S5): Ha egy M típusú csúcs a gráf része, akkor annak vagy bemenetének vagy kimenetének kell lennie legalább egy műveleti egységnek.

(S1) axióma megköveteli, hogy minden termék vagy célként kitűzött feladat szerepeljen a gráfban. Ez a követelmény viszonylag triviális, hiszen minden, ami cél vagy amire az optimalizálás



3.2. ábra. Példa egy P-gráf reprezentációra

történik része kell, hogy legyen a gráfnak. Az (S2) axióma szerint nyersanyag csak akkor lehet egy anyag, ha nem gyártja azt semmi, ellenkező esetben köztes anyagként, vagy termékként kell reprezentálni. (S3) axióma biztosítja, hogy csak a megengedett műveleti egységek jelenhetnek meg a gráfban. (S4) axióma segít leredukálni a hálózatot azokra a műveleti egységekre, amik a cél elérésében szerepet játszhatnak. Amely műveleti egységből nincs út a termékig az felesleges, ezen túl pedig értékes kapacitásokat köthet le megoldáskor. Az (S5) axióma hasonló, mint az (S4), csak anyagok tekintetében. Ha a gráf már redukálásra került az (S4) axióma miatt, úgy csak olyan műveleti egységek szerepelnek benne, amelyek a termékekkel összekötöttek. Amely anyag nincs összeköttetésben ezekkel a műveleti egységekkel, úgy szintén haszontalan a megoldás szempontjából.

Az axiómák segítik a kezdeti struktúra analizálását, és kizárják azokat az anyagokat és műveleti egységeket, amelyek semmilyen módon nem vesznek részt a termék előállításában vagy a cél elérésében. Ezek a tételek továbbá segítenek a potenciális alternatív struktúrák vizsgálatánál a számítási lépések csökkentésében is.

3.2.4. A PNS paraméteres modellje

A matematikai optimalizálási problémák változókból, korlátozásokból és a célfüggvényből állnak. Egy PNS feladatnál a korlátozások tartalmazzák az anyagáramot, a nyersanyagok elérhetőségét, valamint az előállítandó termékmennyiség alsó korlátját annak érdekében, hogy a kereslet kielégítésre kerülhessen. Ezek a korlátozások a következő egyenletekkel fejezhetők ki [8]:

Minden m_j anyagnál Lp_j jelöli a termelés alsó, míg Up_j a termelés felső korlátját. Ha m_j

anyag nem egy termék, akkor

$$Lp_j = \begin{cases} > 0 & \forall m_j \in \mathcal{P} \\ 0 & \textit{egyébként.} \end{cases} \tag{3.8}$$

m_j nyersanyagnál Up_j felső korlát nullára van állítva, egyébként pedig nagyobbnak, vagy egyenlőnek kell lennie, mint Lp_j:

$$Up_j = \begin{cases} 0 & \forall m_j \in \mathcal{R} \\ \geq Lp_j & \textit{egyébként.} \end{cases} \tag{3.9}$$

Az anyagoknál a bruttó fogyasztás felső korlátja is meghatározott. m_j anyag esetén ez Uc_j-vel van jelölve. Nyilvánvaló, hogy nyersanyagok esetében ez több kell, hogy legyen, mint nulla, más esetben viszont meg kell egyeznie nullával.

$$Uc_j = \begin{cases} > 0 & \forall m_j \in \mathcal{R} \\ 0 & \textit{egyébként.} \end{cases} \tag{3.10}$$

Minden köztes anyagnál és mellékterméknél meg kell felelni a mennyiségi egyensúlynak oly módon, hogy minden köztes anyagnál és mellékterméknél az előállított mennyiség nagyobb, vagy egyenlő kell, hogy legyen, mint a fogyasztott mennyiség. A nyersanyagoknál, vagyis azoknál az anyagoknál, amiket felhasznál a rendszer a bemenet negatív arányként jelenik meg. Ezt a költségekkel szorozva negatív költség keletkezne, ami azt eredményezné, hogy a nyersanyagokat minél nagyobb mennyiségben próbálná a modell felhasználni. Ennek elkerüléséhez szükséges az Uc_j változót negatív értékként venni:

$$L_j = \begin{cases} -Uc_j & \forall m_j \in \mathcal{R} \\ Lp_j & \textit{egyébként.} \end{cases} \tag{3.11}$$

és

$$U_j = Up_j. \tag{3.12}$$

Célfüggvényként általában a hálózat bevételeinek maximalizálása vagy a költségek minimalizálása szokott megjelenni, de bármilyen tényezőre megtörténhet az optimalizálás, ha hozzá valamilyen érték társul. Ilyen lehet az, amikor díjazás vagy büntetés kapcsolódik egy-egy tevékenységhez. (Például, ha egységnyi igény kielégítése 10-el növeli a modell értékét, de a közben keletkező $1m^3$ CO_2 3-al csökkenti azt. A mértékegységet a modellező személy adja meg.) Jelen esetben a költségek minimalizálása kerül bemutatásra, ahol a hálózat költsége megegyezik az összes műveleti egység költségének és a nyersanyagok árának összegével. Mivel az optimalizálási modell várhatóan biztosítani fogja a műveleti egységek optimális ellátását az optimális folyamatstruktúra mellett, a költség többek között megadható, mint a mennyiségi terhelés függvénye, például egy fix költségekkel rendelkező lineáris függvényként

$$cf_i + cp_i x_i \tag{3.13}$$

ahol cf_i a fix költség, cp_i pedig egy arányossági konstans, x_i pedig a műveleti egység terhelése, amely jellemzően 0 és 1 között van, azaz 0-100 %. Ha mind a beruházási, mind pedig a működési költség függvényekkel megadott, akkor a költségfüggvény ezeknek a kombinációja lesz.

A fix költségekkel rendelkező lineáris költségfüggvény paraméterei az alábbiak összegéből tevődnek össze: cf_i^{op} és cp_i^{op}, ami a működési költség, valamint $\frac{cf_i^{inv}}{\text{kifizetési időszak}}$ és $\frac{cp_i^{inv}}{\text{kifizetési időszak}}$, ami a beruházás évesített, arányosított költsége:

$$cf_i = \frac{cf_i^{inv}}{\text{kifizetési időszak}} + cf_i^{op} \tag{3.14}$$

$$cp_i = \frac{cp_i^{inv}}{\text{kifizetési időszak}} + cp_i^{op} \tag{3.15}$$

$m_j \in \mathcal{R}$ nyersanyag és $m_j \in \mathcal{P}$ termék árát cm_j jelzi, azon feltételezés mellett, hogy a köztes anyagoknak nincs ára.

m_j anyag és o_i műveleti egység közötti kapcsolat a következő módon adható meg: a_{ji} az m_j o_i általi termelése és fogyasztása közötti különbséget fejezi ki. Ez a következő két kifejezést eredményezi:

$$\alpha_i = \{m_j \in \mathcal{M} : a_{ji} < 0\} \tag{3.16}$$

és

$$\beta_i = \{m_j \in \mathcal{M} : a_{ji} > 0\}. \tag{3.17}$$

$\mathsf{o}^* \subseteq \mathcal{O}$ jelöli az optimális struktúrában szereplő műveleti egységeket, $\mathsf{m}^* \subseteq \mathcal{M}$ pedig az anyagokat, ahol

$$\mathsf{m}^* = \bigcup_{(\alpha_i, \beta_i) \in \mathsf{o}^*} \alpha_i \cup \beta_i. \tag{3.18}$$

Továbbá $\mathbf{x}^* = [x_1, x_2, ..., x_n]^T$ jelöli a műveleti egységek optimális terhelésének vektorát, a célfüggvény pedig az alábbiakkal adható meg:

$$z^* = \sum_{(\alpha_i, \beta_i) = o_i \in \mathsf{o}^*} (cf_i + x_i^*(cp_i - \sum_{m_j \in \alpha \cup \beta} a_{ij} cm_j)). \tag{3.19}$$

A cél a paraméteres PNS-probléma $(\mathcal{P}, \mathcal{R}, \mathcal{O}, \mathbf{A}, \mathbf{cp}, \mathbf{cf}, \mathbf{cm}, \mathbf{u}, \mathbf{L_p}, \mathbf{U_p}, \mathbf{U_c})$ optimális megoldásának megtalálása, azaz a $(\mathsf{m}^*, \mathsf{o}^*, \mathbf{x}^*, z^*)$ hálózat elégítse ki a 3.20-3.24 feltételeket, a mellett, hogy z^* minimális.

$$\forall o_i = (\alpha_i, \beta_i) \in \mathsf{o}^* : m_j \in \alpha_i \iff a_{ji} < 0, m_j \in \beta_i \iff a_{ji} > 0 \tag{3.20}$$

$$\forall m_j \in \mathsf{m}^* \cap \mathcal{R} : -Uc_j \leq \sum_{o_i \in \mathsf{o}^*} a_{ji} x_i^* \leq 0 \tag{3.21}$$

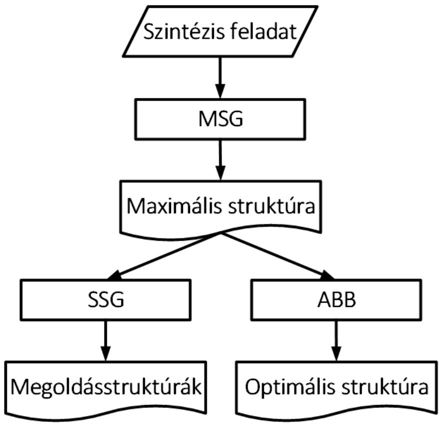
$$\forall m_j \in \mathsf{m}^* \cap \mathcal{P} : Lp_j \leq \sum_{o_i \in \mathsf{o}^*} a_{ji} x_i^* \leq Up_j \tag{3.22}$$

$$\forall m_j \in \mathsf{m}^* \setminus \mathcal{R} \setminus \mathcal{P} : 0 \leq \sum_{o_i \in \mathsf{o}^*} a_{ji} x_i^* \leq Up_j \tag{3.23}$$

$$0 < x_i^* \leq u_i \iff o_i \in \mathsf{o}^* \tag{3.24}$$

$(\mathsf{m}^*, \mathsf{o}^*, \mathbf{x}^*, z^*)$ hálózat lokálisan optimális, akkor és csak akkor, ha a paraméteres PNS feladat $(\mathcal{P}, \mathcal{R}, \mathsf{o}^*, \mathbf{A}, \mathbf{cp}, \mathbf{cf}, \mathbf{cm}, \mathbf{u}, \mathbf{L_p}, \mathbf{U_p}, \mathbf{U_c})$ optimális megoldása. A PNS kombinatorikus algoritmusai, az MSG és SSG az RCABB algoritmus alapjai, amely algoritmus a korlátozás és szétválasztás technikáját alkalmazva képes megoldani a paraméteres PNS problémákat [9].

3.2.5. Algoritmusok



3.3. ábra. A módszertan főbb lépései

Folyamathálózat szintézis esetén a P-gráf keretrendszerben használható három fő kombinatorikus algoritmus az MSG, az SSG és az RCABB. Az **MSG**, vagyis a maximális struktúra generátor (Maximal Structure Generator) biztosítja, hogy a struktúra az axiómáknak megfelelően legyen felépítve. Iteratív módon ellenőrzi a modellt az axiómák szerint és addig redukálja a nem megfelelő részeket, amíg végül a struktúra az összes axiómának meg nem felel. Így távolítja el például azokat a részeket, amik az optimalizálásban nem játszanak szerepet. A szükségtelen részek levágása segít, hogy megoldáskor ne történjen feleslegesen számítás, ez által is gyorsítva az algoritmusok működését. Optimalizálást önmagában nem végez, viszont, mivel ezzel az algoritmussal sokszor jelentős mennyiségű de felesleges számítást lehet elkerülni, ezért a másik két algoritmus futása előtt mindig végrehajtódik, mint egy nulladik lépés. Ez látható a 3.3. ábrán is.

Az **SSG**, azaz a megoldás-struktúra generátor (Solution Structure Generator) előállítja az összes lehetséges megoldásstruktúrát. Csak és kizárólag kombinatorikusan vizsgálja a struktúrákat, a paramétereket még nem veszi figyelembe, vagyis nem végez optimalizálás. Az SSG által generált struktúrák tartalmazzák a legjobb és az N-legjobb megoldás struktúráját is.

Algoritmus 1: RCABB algoritmus

input : (m, o): P-graph, N
output: best_networks

RCABB::Init;
if *is_feasible* **then**
  $solutions := \emptyset \;\; b := \infty$;
  while $subproblems \neq \emptyset$ *and* $(|solutions| < N$ *or there exists subproblem* $(o^-, o^+, o^0, \tilde{z} \in subproblems)$ *such that* $\tilde{z} < b)$ **do**
    $(o^-, o^+, o^0, \tilde{o}, \tilde{x}, \tilde{z}, \text{depth}) :=$ SelectProblem($subproblems$) ;
    $subproblems := subproblems \setminus \{ (o^-, o^+, \tilde{o}, \tilde{x}, \tilde{z}, \text{depth})\}$;
    if $N = 1$ **then**
      o?$=\{o_i : \gamma(o_i, x_i) > 0\}$;
    else
      o?$= \tilde{o}$;
    end
    RCABB::Branch&Bound;
  end
  return *solutions*;
else
  return $\emptyset$;
end

Algoritmus 2: RCABB::Init

N
$subproblems := \emptyset$;
$o^- := \emptyset; o^+ := \emptyset; o^0 := o;\ o^* := \emptyset;\ x^* := 0;\ z^* := \infty\ depth = 0$;
$subproblems := subproblems \cup \{(o^-, o^+, o^0, o^*, x^*, z^*, depth)\}$
(is_feasible, $\tilde{o}$, $\tilde{x}$, $\tilde{z}$) = LP(o^-, o^+, o^0);

A dolgozat szempontjából az egyik legfontosabb algoritmus az **RCABB**. Az ABB (Accelerated Branch & Bound), vagyis a gyorsított korlátozás és szétválasztásos algoritmus már figyelembe veszi a paramétereket is, ezáltal képes az optimalizálásra [9]. Az algoritmus vissza tudja adni nem csak a legjobb, de az N-legjobb megoldást is; többek között ez az egyik erőssége a többi optimalizáló megoldóhoz képest. Az algoritmus az évekkel folyamatosan fejlődött, és a korábban ABB-ként ismert módszert napjainkban már RCABB-ként is ismerik, ahol az "RC" a "Relaxation Controlled"-ot jelenti. Ez az RCABB algoritmus van a grafikus fejlesztői felület, a P-graph Studio mögött is [10].

Az algoritmussal való optimalizálásnál bármilyen célfüggvény használható, ami leírható egy *fix+lineáris* modellel, hiszen az algoritmus a megoldás értékek (value) alapján tudja összehasonlítani az egyik struktúrát a másikkal. Ennek köszönhetően nem csak költségre, de például időre [11] vagy kockázatra is lehet optimalizálni [12].

Algoritmus 3: RCABB::Branch&Bound

if $o^0 = \emptyset$ **then**
  $solutions := solutions \cup \{\ (\tilde{o},\ \tilde{x},\ \tilde{z})\}$;
  if $|solutions| \geq N$ **then**
    comment: update solution set
    $old = solutions$;
    $solutions := \{s = (\tilde{o}, \tilde{x}, \tilde{z}) \in old$ with the 1..N-th smallest values of $\tilde{z}$ $\}$;
    $b = \tilde{z}$ where $s = (\tilde{o}, \tilde{x}, \tilde{z}) \in solutions$ with the N-th smallest value of $\tilde{z}$;
    forall $s = (o^-, o^+, o^0, \tilde{o}, \tilde{x}, \tilde{z}, depth) \in subproblems$ ***where*** $\tilde{z} > b$ **do**
      comment: bounding
      $subproblems := subproblems \backslash s$;
    end
  end
else
  comment: branching
  ô=o_i in o? where γ_i is maximal;
  (is_feasible, $\tilde{o}$, $\tilde{x}$, $\tilde{z}$) = RSGNX(LP($o^- \bigcup$ {ô},o^0/{ô},o^+));
  if *is_feasible* **then**
    subproblems=subproblems $\bigcup$ {($o^- \bigcup$ {ô},o^0/{ô},o^+, $\tilde{o}$, $\tilde{x}$,$\tilde{z}$, depth) }
  end
  (is_feasible, $\tilde{o}$, $\tilde{x}$, $\tilde{z}$) = RSGNX(LP(o^-, o^0 / {ô} , $o^+ \bigcup$ {ô}));
  if *is_feasible* **then**
    subproblems=subproblems $\bigcup$ {(o^-,o^0/{ô},$o^+ \bigcup$ {ô}, $\tilde{o}$, $\tilde{x}$,$\tilde{z}$, depth) }
  end
end

Az RCABB algoritmus minden egyes lépésnél dönt egy műveleti egység bevételéről vagy kizárásáról (1. algoritmus). A különböző alternatív struktúrák a lépésekkel létrejövő keresőfa különböző ágain helyezkednek el, ahol minden csomópont egy részproblémának felel meg a kizárt és bevett műveleti egységek alapján. A gyökértől bármely levél felé haladva minden egyes lépéssel legalább egy műveleti egységgel fog nőni a kizárt vagy bevett műveleti egységek halmaza. Mivel a műveleti egységek száma véges, ezért az algoritmus véges időn belül megáll. Legrosszabb esetben ezeknek a lépéseknek a száma exponenciálisan növekszik, és n darab műveleti egység esetén akár 2^n részhalmaz is keletkezhet.

Az algoritmus akkor áll meg, ha a leveleken megtalálta a felhasználó által kért megoldásszámot. A keresőfa szélessége és magassága a nyitott részproblémáktól függ. Minden pontban az algoritmus egy műveleti egységet bevesz és kizár, ezáltal kétfelé ágaztatva a keresést. Az általános MILP megoldókkal ellentétben itt minden egyes új részproblémánál egy sor logikai követelmény kerül vizsgálatra, például az axiómák alapján [13]. Ezek a követelmények az algoritmusleírásban az RSGNX függvényben kerülnek megvalósításra. Az RSG és NX függvények részletesebben a

[13] és [14] publikációkban találhatók meg. Az RSG függvény kiterjeszti az o^- halmazt a logikai következmények miatt kizárt esetekkel, az NX pedig az o^+ halmazt bővíti ki o^- és o^+ szerint, szintén logikai következmények alapján. Ezekkel az algoritmusokkal csökkenthető a keresési fa mélysége és szélessége anélkül, hogy az ott elhelyezkedő részproblémákat meg kellene oldani. Az RCABB párhuzamosítását és a párhuzamosított algoritmus paraméteroptimalizálását mutatja be a 6. fejezet.

3.3. Alkalmazási területek

Az évek során a PNS és a P-gráf alkalmazási területei folyamatosan bővültek és egyre több publikáció született a témában. Ebben a fejezetben a teljesség igénye nélkül, csak a legjelentősebbek kerülnek bemutatásra, melyek időrendi sorrendben az alábbiak:

- **1979-től: A PNS ötlete**
  A folyamathálózat-szintézis ötlete a hetvenes évek végén fogant meg és 1979-től több előadás formájában is bemutatásra került.
- **1992: Az első publikáció**
  Az első nemzetközi folyóirat publikáció a témában 1992-ben jelent meg. Ekkor még csak kémiai folyamatok ütemezése volt a cél [6].
- **1993: Polinomiális algoritmus a maximális struktúrához**
  Gráfelméleti megközelítésként polinomiális algoritmust alkalmaztak a maximális struktúra generálásához [15].
- **1996: Kombinatorikusan gyorsított B&B**
  Publikálásra került az RCABB elődje, a kombinatorikusan gyorsított korlátozási és szétválasztási algoritmus a PNS-hez [9].
- **2000: Szétválasztási hálózatok, hulladékgazdálkodás és strukturális irányíthatóság**
  2000-ben látott napvilágot az első publikáció, amely a folyamathálózat-szintézist szétválasztási hálózatok optimalizálására alkalmazta a szuperstruktúra kihasználásával [16]. Ugyanebben az évben született meg az első írásos értekezés a PNS hulladékkezelés alkalmazására, valamint a strukturális irányíthatóságra Varga József doktori disszertációjában [17].
- **2001: Hőintegráció**
  Bár előadás már 1999-ben is volt hőintegrációs témakörben a PNS vonatkozásában, mégis 2001-ben került először publikálásra [18].
- **2002: Reakcióút és az első könyvfejezet**
  Már 1997-ben és 1999-ben is volt nemzetközi előadás reakcióutak azonosítása témában, azonban az első írásos publikáció csak 2002-ben látott napvilágot [19]. Ugyanebben az évben került először publikálásra a PNS egy könyvfejezetben [20].

– **2003: Elindíthatóság vizsgálat**
2003-ban a Petri-hálók és a PNS kombinációjával megszületett az első publikáció elindíthatóság vizsgálat témakörben a folyamathálózat-szintézishez kapcsolódóan [21].

– **2010: Szétválasztási hálózat: általános megfogalmazás**
Korábban már több publikáció is érintette, de 2010-ben egy általános metódus került publikálásra a szétválasztási hálózatok témaköréből P-gráf segítségével [22].

– **2011: Jármű-hozzárendelési probléma megoldása PNS-el**
2011-re a PNS alkalmazási területe már széles körben kiterjedt. Bárány és társai ebben az évben publikáltak egy olyan megközelítést, ahol jármű-hozzárendelési problémát oldottak meg hálózatszintézis segítségével a költségek és a környezetre gyakorolt káros hatások csökkentése érdekében [23]. Ez a probléma az ellátási lánc egyik lépését oldja meg, és az ütemezés irányába mutat, fix időpontok bevezetésével pedig már a járműütemezésre is választ ad.

– **2012: Időkorlátos szintézis feladat, evakuálási útvonalak ütemezése, megbízhatóság**
2012-ben megvalósításra került az ütemezési feladatok megoldása PNS-el, amikor is Kalauz és társai publikálták a szintézis feladatok időkorlátokkal történő kibővítését [24]. Ugyanebben az évben jelent meg J.C. Garcia-Ojeda és társainak cikke, melyben a P-gráfot evakuálási útvonalak ütemezésére használták. Utóbbi publikáció jól jelzi annak a spektrumát, hogy milyen különböző területeken sikerült eredményeket elérni PNS-el [25]. Bár előadások korábban is voltak, az első publikáció a folyamathálózatok megbízhatóságáról az ellátási láncok esetében szintén ekkor látott napvilágot [26].

– **2014: Ütemezés P-gráffal és a multiperiódusos P-gráf modellek ötlete**
A folyamatok ütemezésére 2014-ben időkorlátos folyamathálózatokat alkalmaztak Frits és társai. Egy kellően általános modellt sikerült bevezetni, ami egyik alapját képezi jelen dolgozat egy fejezetének is [27]. Ugyanebben az évben született az ötlet a multiperiódusos P-gráfok bevezetésére, bár ekkor még csak egy egyszerű esettanulmányon keresztül valósult meg a több periódusos jelleg [28].

– **2015: Szoftver a modellezéshez és megoldáshoz; az RCABB algoritmus párhuzamosítása**
2015-ben két korábbi, kezdetleges, de már felhasználói felülettel rendelkező szoftver alapján implementálásra került a P-graph Studio, ahol grafikusan lehetett modellezni, és az elkészült struktúrákat megoldani [10]. (Részletesebben a 3.4. fejezet foglalkozik vele.) Szintén ebben az évben került implementálásra az RCABB algoritmus egy modernebb megközelítésű párhuzamosítása, amit később a paraméteroptimalizálás egészített ki [29]. (Ez utóbbi a 6. fejezetben kerül bemutatásra.)

– **2016: P-gráf modell energiaelosztásra**
Aviso és társai 2016-ban publikáltak egy tanulmányt, ahol energiaellátást valósítottak meg

multiperiódusos P-gráf modellel, izolált rendszereknél [30]. (Ez a publikáció a 4. fejezethez kapcsolódik.)

– **2017: Általános áttekintés**
  Varbanov, Friedler és Klemes professzor urak 2017-ben egy publikációban összegezték a P-gráffal az elmúlt évtizedekben elért eredményeket. A cikk jól összefoglalja, hogy honnan indult és hova fejlődött a módszertan, ez által egy áttekintést adva a P-gráf múltjáról, jelenéről és jövőjéről [31].

– **2018: Teljes telephely-rendszer tervezése P-gráf segítségével és különböző P-graph Studio fejlesztések**
  2018-ban Walmsley és társai energiaellátó rendszert, vagyis hő- és áramellátást optimalizáltak PNS-el [32]. (Ez a publikáció a 4. fejezethez kapcsolódik.) Ebben az évben került továbbfejlesztésre és kiegészítésre a P-graph Studio a multiperiódusos modellek automatizálásával, illetve azzal, hogy hulladékgazdálkodási szempontokat is képes legyen figyelembe venni [33].

3.3.1. Ütemezés TCPNS-el

Az elmúlt két évtizedben a szakaszos folyamatok ütemezése egyre inkább egy aktívan kutatott területté vált, és számos eredmény született a témában. Mint a többi ütemezési feladatnál, itt is az a cél, hogy feladatok és berendezések egymáshoz rendelése optimális legyen. A gyakorlatban a legtöbb ilyen ütemezési feladatnál a folyamatok szekvenciális sorrendben vannak, de még ezeknél a szekvenciális folyamatoknál is lehet többtermékes és többcélú (multipurpose) modellekről beszélni. A többtermékes folyamatoknál minden termékhez ugyanaz a szekvenciális recept tartozik, míg a többcélúnál minden terméket ugyanazon lépések különböző szekvenciáival lehet létrehozni. A leggyakrabban vagy a gyártási időt kell minimalizálni, vagy a teljesítmény, nyereség maximalizálására kell törekedni.

A klasszikus ütemezési probléma kiterjesztése az MPTS (Malleable Parallel Task Scheduling), ahol párhuzamosan több feldolgozó is képes kezelni egy feladatot. Bár ez a rugalmasság a teljes feldolgozási, gyártási időt lecsökkentené, de ezzel együtt nagyban megnehezíti a probléma megoldhatóságát. Frits és Bertók [27] 2014-ben bemutattak egy új megközelítést, ahol az ütemezési problémákat folyamathálózat-szintézis (PNS) problémaként fogalmazták meg, illetve lehetővé tették az automatikus modellgenerálást vegyes tárolási stratégiát követve. Általános megoldást adtak az MPTS problémákra, mely továbbá garantálja az optimumot is. Ez volt az időkorlátos, vagyis *Time Constrained PNS* (TCPNS).

Ehhez a szuperstruktúrát a TCPNS-nek megfelelően az ütemezés recept gráfját alapul véve kell létrehozni, ahol az egyes műveleti egységek mint erőforrások fognak megjelenni. A cél egy olyan folyamathálózat létrehozása, ami kielégíti a célokat miközben figyelembe veszi a feladatok kapcsolatát. A szuperstruktúra létrehozása három lépésből áll:

– Alaptevékenységek meghatározása: Az alaptevékenységek közé tartoznak a berendezések feladatokhoz való rendelése, az input betöltése a berendezésbe, valamint a végrehajtás.

– Recept élek gráfhoz rendelése: Az élek határozzák meg a sorrendet. A következő berendezés csak akkor indulhat el, ha az azt megelőző befejezte a munkát.

– Tárolási politikától függően új állapotok bevezetése: A tárolási politika függvényében új állapotokat kell bevezetni arra, hogy a tárolás az elvártaknak megfelelően történhessen. Például biztosítani kell, hogy egy berendezés ne tudjon új feladatot vállalni addig, amíg az átvitel meg nem történt. Ezt különböző előfeltételek bevezetésével lehet biztosítani.

Az eredményeik kimutatták, hogy amennyiben az ütemezési problémák időkorlátos PNS feladatként kerülnek modellezésre, úgy utóbbi keretrendszer számos előnyét ki lehet használni a megoldás érdekében. Az általános megfogalmazás miatt ez a matematikai modell automatikusan generálható akár vegyes tárolási stratégia esetén is, anélkül, hogy szükség lenne időintervallumok bevezetésére. A megoldás végén az optimum garantált. A gyártósor kiegyensúlyozás, vagy más néven Line Balancing egy ehhez hasonló, de egyszerűbben modellezhető probléma, amivel a 5. fejezet foglalkozik részletesebben.

3.3.2. Multiperiódusos modellek

Multiperiódusos problémák számos helyen előfordulnak a való életben. Ezek olyan esetek, ahol valamilyen folyamatosság figyelhető meg, és az egymást követő periódusok nem függetlenek egymástól. Ez a folyamatosság fellelhető a mezőgazdaságban, ahol a termények folyamatosan érnek, de akár egy összeszerelő üzemnél is, ahol két nap között képesek tárolni a félkész termékeket, amik végül hatással lesznek a következő napi termelésre is. Mivel számos helyen megjelenik a multiperiódusos jelleg, ezért az optimalizálás területén is egyre nagyobb figyelmet kezd kapni.

Heckl, Halász, Szlama, Cabezas és Friedler 2014-ben publikálták az első olyan multiperiódusos megközelítést, ahol P-gráfot használtak az ütemezésre. Esettanulmányukban egy almapucoló üzem éves működését modellezték [34]. A kiinduló probléma szerint az üzem évente 30 tonna almát pucolt meg, ami havi 2,5 tonna terhelést jelentett, vagyis elméletileg elég lett volna egy 2,5 tonna/hónap kapacitású hámozógép alkalmazása. A valóságban azonban az alma érése főleg két hónapra datálható, emiatt szükséges volt a modell felbontása részekre, amely részek strukturálisan bár nagyon hasonlóak, paraméterezésüket tekintve eltérőek. Szükséges volt továbbá megoldani a beruházási költség szétosztását a periódusok között azok hosszával egyenes arányú súlyozással, és a fix költséget is ennek megfelelően arányosítani. Eredményeikben megmutatták, hogy az eredeti struktúránál a valóság sokkal költségesebb, hiszen egy sokkal nagyobb kapacitású gép szükséges a feladat elvégzéséhez. Munkájukkal felhívták a figyelmet a többperiódusú modellekre és az annak megfelelő, pontos paraméterezésre. Publikációjuk és megközelítésük a témában úttörő volt, ámbár eredményeik csak esettanulmányokhoz kapcsolódtak, és nem adtak egy általános modell-leírást a többperiódusú problémák P-gráffal történő modellezéséhez. Ennek hozományaként készült el, és a dolgozat részét képezi (4. fejezet) egy olyan általános leírás, ami bármely, többperiódusú feladatnál segít a modell megvalósításában.

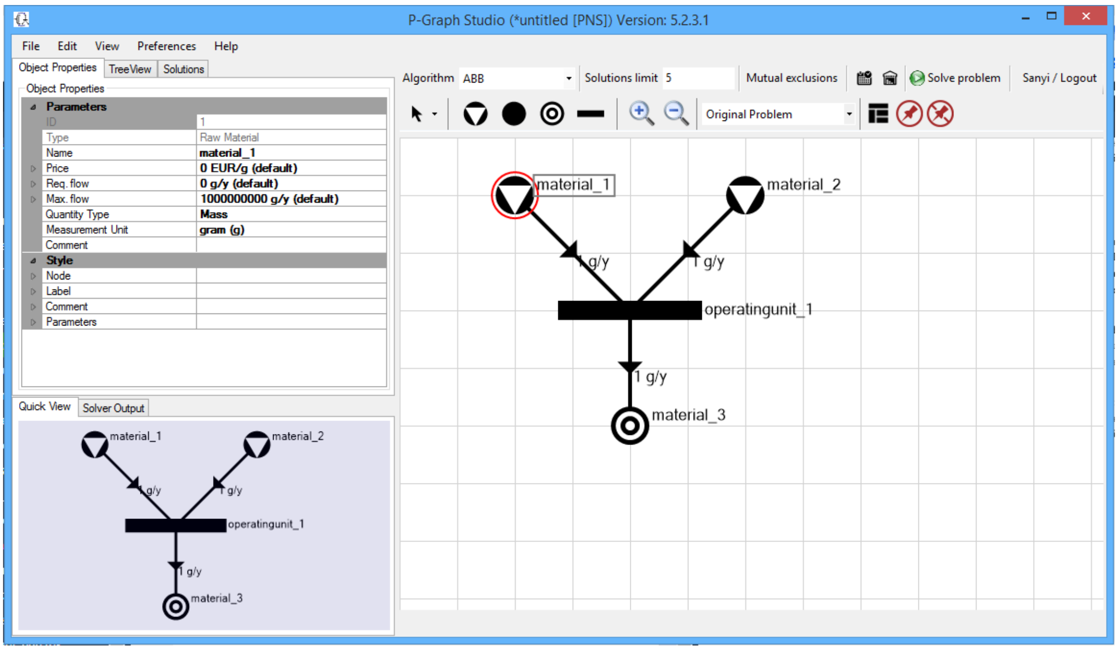
3.4. Szoftver

A P-gráfos modellek megoldásához kezdetben nyílt, felhasználói felülettel nem rendelkező programok és algoritmusok álltak rendelkezésre. A 2010-es évek elején kerültek először implementálásra majd mindenki számára szabadon elérhetővé olyan szoftverek, amik már grafikus felülettel is rendelkeztek. Ekkor még két külön applikáció létezett a P-gráf grafikus rajzolására, illetve annak megoldására:

- A *PNS Draw* a napjainkban elérhető szoftver, a P-graph Studio felületéhez nagyon hasonló, grafikus felületet kínált a P-gráf modell szerkesztéséhez. A "fogd és vidd" felületen egyszerűen lehetett egy P-gráfot grafikusan felrajzolni [35].

- A *PNS Studio* a modellek megoldásában nyújtott segítséget. A modellt itt is fel lehetett vinni, de csak az egyes elemek és tulajdonságainak megadásával, vagyis nem gráf formátumban [36]. A két szoftver között volt átjárhatóság, de ezt csak körülményesen lehetett megoldani.

2015-ben a két szoftver alapján egy ingyenesen elérhető, C# alapú asztali alkalmazás került lefejlesztésre $P - graphStudio$ néven a Pannon Egyetem Rendszer- és Számítástudományi Tanszékén [10]. A szoftver alkalmas a grafikus modellezésre, annak felparaméterezésére, megoldására. A 3.4. ábrán látható a megoldó felülete, ahol a *PNS Draw*-hoz hasonló módon, "fogd és vidd" módszerrel lehet összeállítani egy P-gráf modellt.



3.4. ábra. A P-graph Studio felülete

Az egyes elemek paraméterbeállítása a bal oldalon történik, jobb oldalon pedig maga a modell található. Az állapotsoron legördülő menüből lehet kiválasztani a szükséges algoritmust, és szintén itt lehet megadni a kívánt megoldásszámot, valamint a kölcsönösen kizáró műveleti egységeket. Megoldás után a megoldások között lehet váltani a nézeten, a megoldások tulajdonságai részletesen pedig a baloldalon találhatóak meg. A "Preferences" menüpont alatt számos beállítási lehetőség található. Ilyen például az is, hogy a megoldás során keletkező plusz állományokat ne törölje a rendszer a megoldás után. Ezek a P-gráf megoldónak a nyers ki- és bemenetei, amik kompatibilisek a korábbi modellformátummal és megoldókkal. A modellben szereplő elemek alapértelmezett beállításain érdemes változtatni (szintén a beállítások alatt), ha több, hasonló tulajdonságú elem is szerepel a modellben. (Például, ha minden műveleti egység felső korlátja 10 darab.) A szoftver lehetőséget kínál a kinézet formázására, és az eredmények elmentésére is számos lehetőséget ad. A grafikus (raszteres és vektorgrafikus) exportáláson túl lehetséges vele táblázatos formában kimenteni a modellt, egységesítve, vagy részletesen, a megoldásokkal együtt. Lehetséges vele $ZIMPL$ formátumba is exportálni, ami egy olyan, egyszerű nyelv, amiből könnyedén lehet az általános LP és MILP megoldók formátumába exportálni a problémát [37, 38]. Ez a különböző megoldók összehasonlításánál játszhat szerepet. A P-graph Studio a legújabb fejlesztések és hibajavítások tekintetében naprakész, megoldhatók vele a PNS-en kívül, TCPNS feladatok is, valamint az ebben a dolgozatban szereplő multiperiódusos és párhuzamosított részeket is tartalmazza.

3.5. Megoldó algoritmusok párhuzamosításai

A modellek optimalizálása fontos a hatékonyság miatt, azonban nem elég magát a módszert optimalizálni, programozástechnikailag is figyelni kell az erőforrások minél jobb kihasználására. A számítógépek teljesítménynövekedésével és a többmagos megoldásokkal ma már pazarlás lenne ezeket a tulajdonságokat nem kihasználni egy komplex, nagy számításigényű algoritmus esetében. Ha a probléma vegyes egész matematikai programozás lineáris (MILP), vagy nem lineáris (MINLP) feladataként van definiálva, az általános megoldók csupán egy, a legjobb eredményt adják vissza [8]. A P-gráf megoldónál az egyik legfontosabb és legtöbbet használt algoritmus az RCABB (Relaxation Controlled Accelerated Branch and Bound), amely ezekkel ellentétben nem csupán a legjobb, de az N-legjobb megoldást is képes visszaadni. A következőkben ismertetésre kerülnek a Branch and Bound algoritmus párhuzamosíthatóságának lehetőségei, valamint a P-gráf megoldó RCABB algoritmusának korábban publikálásra került párhuzamosításának megvalósítása.

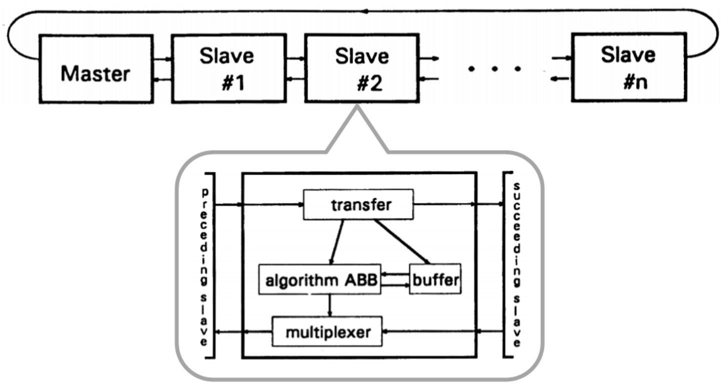
3.5.1. Branch & Bound algoritmusok párhuzamosítása

A Branch-and-Bound (vagyis korlátozás és szétválasztás) típusú algoritmusok párhuzamosításához számos ajánlás került már publikálásra. A kereső algoritmus előrehaladását a legegyszerűbben egy keresőfával lehet illusztrálni, ahol az elágazások a különböző alternatív döntések, és e döntések kombinációi a részfák. A keresési algoritmusok párhuzamos implementációiban a szálak vagy külön számítógépeken, processzorokon, vagy processzormagokon futnak, és jellemzően a fent említett keresőfának egy-egy részfáját oldják meg [39, 40, 41]. A két fő osztálya ezeknek az algoritmusoknak az elosztott és a közös memóriát használó rendszerek. A közös memóriájú rendszereknél minden elérhető számítási egység memóriája fizikailag közös, és ezt a memóriát mindegyik szál eléri [42, 43]. Az elosztott memóriájú rendszereknél minden szál a saját dedikált memóriáját irányítja, ami nem érhető el a többi szál számára [44].

Az elosztott rendszerek egyik közös jellemzője, hogy létezik egy úgynevezett "mester" szál, amelyik összehangolja a szálak működését [45, 46, 47]. A "szolgák", vagyis a többi szál csak ezzel a mester szállal kommunikálhatnak, egymással nem [48, 49, 41]. A mester-szolga topológia mellett egy másik népszerű párhuzamosítási megoldás a "kollegiális" rendszer [50]. Ezeknek a rendszereknek egy fontos jellemzője a közös memória, amely globális változókat tartalmaz. Globális változók lehetnek a megoldások, a részmegoldások és a részfák, amik minden szál részére egyformán elérhetőek [51, 41, 40]. Ezek a megoldások általában CPU magokon párhuzamosítanak, de napjainkban egyre nagyobb tért hódít a GPU-n történő párhuzamosítás is [52, 39].

3.5.2. Párhuzamosított RCABB algoritmus

Az RCABB algoritmus korábbi változatának, az ABB algoritmusnak egy párhuzamosított változatát már 1995-ben elkészítették Varga és társai [53]. Az akkori megoldás a mester-szolga elven, több processzoron működött. A párhuzamosított számítás megvalósításának topológiája azokban az időkben jelentősen limitálta az információáramlás sebességét. A publikált megoldásban a szolgák gyűrű topológiában helyezkedtek el, és ebben a gyűrűben kapott helyet a mester is. Ez a 3.5. ábrán látható.



3.5. ábra. Az ABB algoritmus mester-szolga implementációja gyűrű topológiával

Minden blokkban a transzfer és a multiplexer üzeneteket küldhet és kaphat. Könnyű belátni, hogy a mestertől az utolsó blokkba kézbesített üzenet jut el a leglassabban, mivel az információ az összes köztes blokkon át kell, hogy haladjon, ahhoz, hogy elérjen az utolsóig. Hasonlóképp, ha egy középen elhelyezkedő szolga választ akar küldeni a mesternek, az üzenetnek hosszú utat kell bejárnia, hiszen egy -a közvetlenül mester mellett lévő- szolgát leszámítva egyik sem kapcsolódik közvetlenül a mesterhez. Továbbá elmondható, hogy a mester egyetlen feladata az, hogy irányítsa a szolgák munkáját.

3.5.3. Párhuzamosítási lehetőségek napjainkban

Az elmúlt évtizedekben egyre több és több mag jelent meg a processzorokban, amik képesek elérni egy közös memóriát. Ez lehetővé teszi egy olyan párhuzamosított algoritmus megvalósítását, amely rugalmasabb, és hatékonyabban, kiegyenlítettebben képes a szálak terhelését megoldani. A fő kérdés, ami felmerül a párhuzamosítással kapcsolatban, hogy hogyan lehet a leginkább kiegyensúlyozott a szálak terheltsége amellett, hogy az egymás közti kommunikáció ideje a lehető legkevesebb legyen. A gyakori kommunikáció lassítja az algoritmust, de, ha a kommunikáció ritkábban valósul meg, előfordulhat, hogy a szálak felesleges számításokat végeznek az információhiány miatt. Ez a következők miatt történhet: Az úgynevezett vágás vagy korlátozás segít abban, hogy a biztosan nem jó részproblémákat figyelmen kívül lehessen hagyni számítás közben, például, ha egy részproblémáról már megoldása előtt lehet tudni, hogy rosszabb értéket fog adni, mint egy már megtalált megoldás, akkor felesleges megoldani azt. Ha viszont az információ frissítési gyakoriság alacsony, elképzelhető, hogy egy szál megtalál egy olyan megoldást, ami miatt a többi szál figyelmen kívül kéne, hogy hagyjon részproblémákat, de mivel ez az információ még nem ért el hozzájuk, ezért ezeket a felesleges részproblémákat is elkezdik megoldani, ami szükségtelen idővesztességgel jár. A 6. fejezetben bemutatásra kerül az RCABB algoritmus több magon megvalósított változata, valamint a párhuzamosítás paramétereinek optimalizálása annak érdekében, hogy egy PNS feladatot a lehető leghatékonyabban legyen képes megoldani.

4. fejezet

Multiperiódusos P-gráf

Ahogy az már a 3. fejezetben is említésre került, a P-gráf módszertan az elmúlt években nyitott a többperiódusos feladatok megoldása felé. Elsőként Heckl és társai publikáltak multiperiódusos P-gráfról, majd az évek során egyre több cikk született a témában, de ezek csak esettanulmányok voltak speciális esetekre, nem pedig általánosan használható modellek és modell-leírások. A következőkben ismertetésre kerülnek a multiperiódusos P-gráfhoz kapcsolódó új kutatási eredményeim.

4.1. Kapacitáskiegyenlítés tárolók bevezetésével

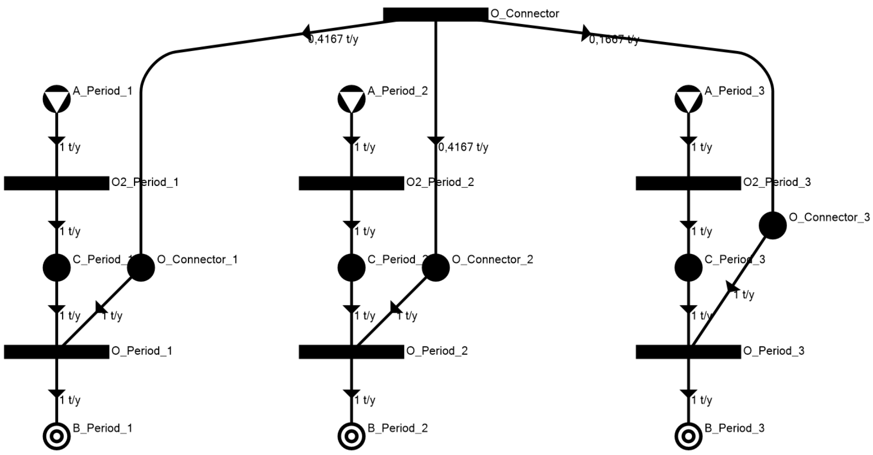
Heckl és társai a 2014-ben publikált multiperiódusos P-gráfról szóló cikkükben egy almapucoló üzem példáján keresztül mutatták be a periódusokra bontás relevanciáját. Az eredeti, nem periódusos modell szerint az üzem 30 tonna almát pucol meg évente -vagyis havonta 2,5 tonnát-, ezért elegendő egy 2,5 tonna/hónap kapacitású hámozó berendezést venni. A valóságban azonban az alma főként nyár végén és ősszel érik, ezért ott megnövekedett kapacitással kell számolni. A 4.1. táblázat alapján jól látható, hogy az eredetileg 170 €-val számolt modell költsége valójában 290 €. A 4.1. ábra reprezentálja a modell P-gráfos megvalósítását. Elmondható tehát, hogy a periódusokra bontással a valóságot sokkal jobban leíró modell adható meg.

A példából jól látszik, hogy a nem egyenletes terhelés jelentős mértékben és negatívan hat a költségekre, ezért felmerült a kérdés, hogy hogyan lehet ezt valamilyen módon kiegyenlíteni. A megoldást a periódusok közötti tárolási lehetőségek adják, amiknek köszönhetően az anyagok egy részét a következő periódusra lehet átvinni. Tárolónak minősül például egy pincehelyiség, hűtőház, raktár vagy akár az összeszerelő szalag, azaz minden olyan hely, ahol két periódus között az anyag elhelyezhető, tárolható, vagy utóbbi esetben otthagyható. Ez az anyag lehet nyersanyag, köztes anyag vagy akár termék is. Nyersanyag és köztes anyagok tárolásánál a feldolgozást lehet kiegyenlíteni a periódusok között.

A tárolással nem csak a munkavégzést lehet egyenletesebbé tenni, de akár piaci befolyásolás is lehetséges a kínálat elosztásán keresztül. A fenti példából kiindulva, ha a megpucolt almákat nem

	Eredeti modell	Multiperiódusos modell		
Periódus:	1.	1.	2.	3.
Periódus hossza (év)	1	5/12	5/12	2/12
Havi terhelés (t/hó)	2,5	1	2	7,5
Periódusos terhelés (t/per.)	30	5	10	15
Aktuális kapacitás (t/év)	30	12	24	90
Maximális kapacitás (t/év)	30		90	
Kapacitás költség (€/per.)	96	17,5	32,5	46
Összes költség (€/év)	170		290	

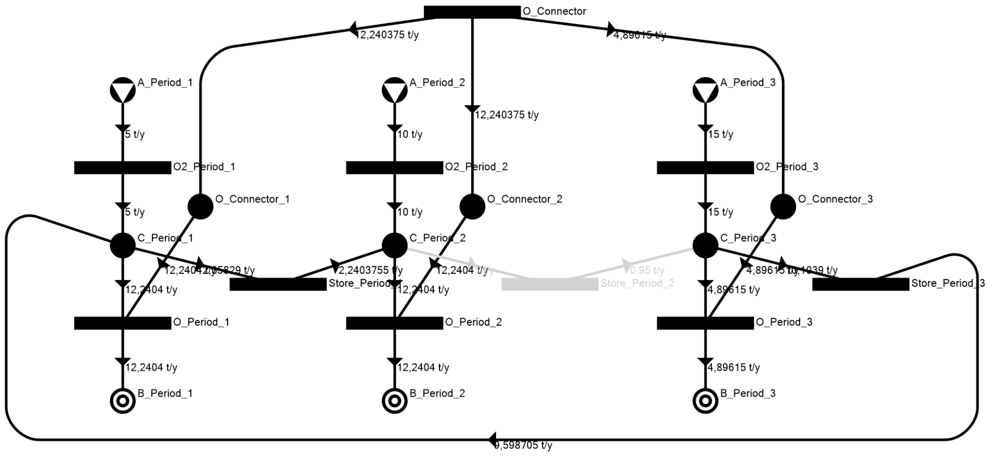
4.1. táblázat. Az almapucoló üzem egy és többperiódusú adatai



4.1. ábra. Az almapucolós példa három periódusú modellje

a gyártásukkor próbálják meg értékesíteni, hanem azokat tárolva, egy későbbi időpontban küldik a piacra, amikor annak az ára már magasabb, mint a főszezonban, úgy több profitot érhetnek el. Természetesen a tárolók esetében is számolni kell különböző költségekkel. Egy tárolónál -ahogy általában az összes műveleti egységnél - felmerülhet fix, arányos, illetve beruházási költség is. Fix költség esetén a tároló használata után egy fix összeg fizetendő, és ez független a tárolt anyagok mennyiségétől. Arányos költség esetében a fizetendő összeg a tárolt anyag mennyiségével arányos. Amennyiben a tárolót létre kell hozni - például felépíteni egy raktárat- úgy annak beruházási költségei lehetnek, amik a kifizetési időszakban leosztásra kerülnek. Veszteségként jelenik meg továbbá az amortizáció is. Tárolás során felmerülhet a kérdés, hogy az anyag, amit tárolni kell egyáltalán eltárolható-e, vagyis tulajdonságait teljes mértékben megtartva kerül ki a tárolóból a következő periódus elején, vagy sem. Az almának egy része megrohadhat a tárolás alatt, ahogyan az akkumulátorok töltöttsége is csökken az idővel. Ezt a fajta veszteséget is figyelembe kell venni a tervezéskor, hogy még pontosabb modell szülessen.

A tárolókat az egyes periódusok között műveleti egységek reprezentálják. A 4.2. ábrán a korábbi példa tárolókkal történő kiterjesztése látható, illetve annak is a legjobb megoldása. Ebben



4.2. ábra. Az almapucolós példa kiegészítése tárolóval

a példában a tároló már adott volt, vagyis nem volt szükség beruházási költségre, továbbá a periódusok -ahogyan az évszakok is- ismétlődnek, így a harmadik periódus tárolója az első periódusba csatlakozik vissza. A tárolóban tárolt almák 5%-a megy tönkre, azaz a tárolókból kijövő élek súlya 0,95 lesz. Látható, hogy a tárolók segítségével megvalósítható az arányosan elosztott végrehajtás, és ekkor már elegendő lesz egy kisebb kapacitású hámozógép is. Az már a döntéshozók feladata, hogy mérlegeljék, az hoz-e több hasznot, ha az almák 100%-át képesek eladni, vagy ha a nagyobb kapacitású gép helyett a kisebbet veszik meg.

4.2. Multiperiódusos P-gráf modell általános leírása

A multiperiódusos P-gráfhoz korábban csak esettanulmányok születtek, de nem létezett egy kellően általános leírás, ami akár egy szoftveres implementáció alapja is lehet. Ahhoz, hogy egy nem periodikus modellből multiperiódusost lehessen előállítani, az alábbi plusz információkra van szükség:

- **Periódusok száma**: A periódusok száma adja meg, hogy hány periódusból fog állni a modell.
- **Periódusok nevei**: Minden egyes periódushoz tartozik egy egyedi azonosító, vagy név. Ez csupán a beazonosításra szolgál, a modellmegoldásban nem játszik szerepet. Az egyes periódusokra bontásban az adattagok ezzel az azonosítóval kerülnek kiegészítésre, így a későbbiekben akár szoftveresen is könnyebb lesz a végeredmény, azaz a megoldott modell értelmezése.
- **Periódusok hosszai**: Ahogy az almapucolós példán is látszott, nem feltétlenül egyforma hosszúak az egyes periódusok. Ez a költségek leosztásában játszhat szerepet. Mivel modellezési szempontból az egyes periódushosszak egymáshoz viszonyított aránya a lényeges,

ezért külön mértékegysége nincs.

– **Anyagok tulajdonságai az egyes periódusokban**: Az eredeti, nem periódusos modell mellett megadható minden anyagra, hogy az egyes periódusokban milyen értéket vegyen fel az áruk és a kötelező mennyiségük minimum, valamint maximum értéke. Amennyiben ez nem kerül megadásra, úgy az eredeti modell szerinti értéket kapják.

– **Kiterjesztés vagy felbontás?**: Nem mindegy, hogy az eredeti modellt kell felbontani több periódusra, avagy az eredeti modell egy periódust reprezentál, amit "többszörözni" kell. Strukturálisan az eredmény ugyanaz, viszont előbbi esetben a műveleti egységeken megjelenő költségeket arányosítani kell a periódusok hosszának megfelelően, míg utóbbi esetben erre nincs szükség.

– **Kivételek**: Lehetnek olyan anyagok vagy műveleti egységek amiket nem lehet több periódusra bontani. Ilyen lehet az, amikor a teljes periódusra összességében van megadva egy nyersanyag mennyisége, de az már lényegtelen, hogy az egyes periódusokban milyen mértékben kerül felhasználásra, ezért lehetőséget kell biztosítani arra, hogy az ilyen kivételek külön megadhatóak legyenek.

Multiperiódusos feladatoknál a megfelelő módosításokat a megoldás előtt kell végrehajtani, hiszen ezek csupán a struktúrát és annak paramétereit érintik, a megoldásuk menete viszont megegyezik egy nem-periódusos modell megoldásával. A struktúra létrehozásának általános leírását adja a 4. algoritmusnál látható pszeudokód. A könnyebb átláthatóság miatt ez három fő részre lett bontva.

Algoritmus 4: Multiperiódusos szétbontás

input : Alap P-gráf, Periódusok tulajdonságai, Multiperiódusos adatok, Kivételek
output: Multiperiódusos struktúra

Masolas_Beillesztes ;
Beruhazasi_koltseg_kivezetes;
Kivetelek_es_Torles;

Az első fő lépésben a másolás és beillesztés játssza a fő szerepet (5. algoritmus). Kezdeti lépésként meg kell határozni, hogy melyik az a struktúrarész amit a multiperiódusosság érint. Ez minden olyan anyagra, műveleti egységre és élre igaz lesz, ami nem került korábban a kivételek közé. Mivel a modellalkotás közben a meglévő struktúra egy része törlődik, szükséges tudni, hogy melyek azok a műveleti egységek, amik a periódusos részhez tartoznak, és rendelkeznek beruházási költséggel. Ezek az elemek egy külön halmazba kerülnek gyűjtésre.

Ezt követően a periódusszámnak megfelelő alkalommal a másolásra kijelölt struktúrát hozzá kell adni a P-gráfhoz, figyelve arra, hogy ha az anyagnak volt egyéni, periódusos tulajdonság megadva, úgy azokat kapja meg. Figyelmet kell még fordítani továbbá arra, hogy a periódusos műveleti egységeknek volt-e fix költségük. Amennyiben igen, és a modellt nem "többszörözni" kell, hanem részekre bontani (ahogyan az almapucoló példában is volt), úgy ezt a fix költséget arányosítani kell a műveleti egységeken a periódusok hosszával arányosan. A műveleti egységek beruházási költsége külön kerül majd reprezentálásra, így az minden esetben '0' lesz a most beillesztett elemeknél. Mind anyagnál, mind pedig műveleti egységnél figyelni kell arra, hogy egyéni, jól megkülönböztethető azonosítóval - P-gráf esetén névvel - legyenek ellátva az elemek. Ez általában az *eredeti elem neve* + "_Periodus_" + *periódus neve*, mivel ebből az eredeti elemet és a periódust is vissza lehet fejteni.

Algoritmus 5: Masolas_Beillesztes

```
masolandoStruktura := {M \ M^Ex, O \ O^Ex, E \ E^Ex};
szumPeriodusHossz := Σ_{i=1}^{|Per|} per_i → hossz;
foreach o_i ∈ O \ O^Ex do
    if o_i.beruhazasiKoltseg > 0 then
        O_beruh.hozzaad(o_i.masolat());
    end
end
foreach per_i ∈ Per do
    beillesztettStruktura = P − graf.hozzaad(masolandoStruktura.masolat()) ;
    foreach elem ∈ beillesztettStruktura do
        if elem → tipus = anyag then
            if per_i.vanEgyeniAnyagtulajdonsag(elem) then
                elem.tulajdonsagok = per_i.anyagTulajdonsagok(elem);
            end
            elem → nev = elem → nev + "_Periodus_" + per_i → periodusnev;
        end
        if elem → tipus = muveletiegyseg then
            elem.beruhazasiKoltseg = 0;
            if elem → fixkoltseg > 0 then
                if felbontas then
                    elem → fixkoltseg = (elem → fixkoltseg/szumPeriodusHossz) ∗
                      ∗ per_i → hossz;

                end
            end
            elem → nev = elem → nev + "_Periodus_" + per_i → periodusnev;
        end
    end
end
```

A második fő lépésben a beruházási költségek kivezetését kell megvalósítani (6. algoritmus). Ez azért szükséges, mert a beruházás csak egyszer kell, hogy megvalósuljon, az onnantól érvényes lesz az összes periódusban is. Ez egy fajta függőséget is reprezentál, vagyis, ha nem történik beruházás, úgy a periódusokban sem lesz használható a műveleti egység. Tekinthető úgy is, mint fizikai megvalósítás, a periódusokban pedig a használat jelenik meg. Ebben a lépésben a korábban kigyűjtött, beruházási költségekkel rendelkező műveleti egységeken kell végigiterálni, és mindegyiknél jelölni, hogy egy kivezetett, periódusokon kívül megjelenő elemről van szó, majd ezt hozzáadni a gráfhoz. Ezt a műveleti egységet össze kell kötni köztes anyagokon a keresztül az összes, periódusokban megtalálható használattal. Itt is figyelni kell a súlyozásra, vagyis, hogy a periódusok hosszával arányos beruházási költség terhelődjön az egyes periódusos használatokra.

Ez, a kivezetett műveleti egységet az újonnan létrehozott köztes anyagokkal összekötő él súlyán jelenik meg. Az új elemeket természetesen hozzá kell adni a gráfhoz.

Algoritmus 6: Beruhazasi_koltseg_kivezetes

foreach $o_i \in O_{beruh}$ **do**
  $o_i \rightarrow nev = o_i \rightarrow nev + "\_Eloszto"$;
  $P - graf.hozzaad(o_i)$;
  foreach $per_i \in Per$ **do**
    $ujAnyag \rightarrow nev = o_i \rightarrow nev + "\_Eloszto\_" + per_i \rightarrow nev$;
    $ujEl1(o_i, ujAnyag)$;
    $ujEl1 \rightarrow suly = per_i \rightarrow hossz/szumPeriodusHossz$;
    $ujEl2(ujanyag, O.megtalal(o \rightarrow nev == o_i \rightarrow nev + "\_Periodus\_" + per_i \rightarrow nev))$;
    $P - graf.hozzaad(ujAnyag)$;
    $P - graf.hozzaad(ujEl1)$;
    $P - graf.hozzaad(ujEl2)$;
  end
end

A harmadik fő lépésben a kivételek bekötését kell megvalósítani. Itt elegendő végigiterálni a kivételekhez tartozó éleken, és megvizsgálni, hogy a kezdő vagy végpontjuk a nem kivétel osztályhoz tartozik-e. Ez négyféleképpen valósulhat meg:

- A kezdőpontja olyan anyag, ami nem kivétel
- A kezdőpontja olyan műveleti egység, ami nem kivétel
- A végpontja olyan anyag, ami nem kivétel
- A végpontja olyan műveleti egység, ami nem kivétel

Ezeket az éleket meg kell valósítani minden egyes periódusra is. Tehát, ha eddig volt egy él, aminek az egyik végpontja periódusos, míg másik végpontja a kivételekhez tartozó elem, úgy a kivételekhez tartozó elemből minden egyes periódusba vezetni kell egy-egy élt. A gráfhoz ezeket az új éleket hozzá kell adni, ezzel egy időben pedig az eredetit törölni. Zárásként pedig a másolandó struktúrát, vagyis az eredeti gráfból azokat az elemeket, amik nem kivételek törölni kell (7. algoritmus). Az algoritmus végrehajtása után előáll a már periódusokra bontott P-gráf modell.

4.3. A tárolók megvalósítása algoritmikusan

Ahogy a fejezet elején szó volt róla, a tárolók segíthetnek abban hogy az anyagfelhasználás és gyártás kiegyenlítettebb lehessen. Fontos, hogy legyen egy általános leírás a tárolók bevezetésére is, viszont egy kellően általános tároló számos kérdést vet fel: Milyen anyagot lehet benne tárolni? Egy vagy többféle anyag tárolására is képes? Van esetleg beruházási költsége? Mekkora a kapacitása? A kapacitásból egy egység tárolt anyag mennyit foglal el? Minden periódus között képes tárolni? Tároláskor amortizálódik a termék? ...stb.

Algoritmus 7: Kivetelek_es_Torles

```
foreach e_i ∈ E^Ex do
    foreach per_i ∈ Per do
        if (e_i → honnan → tipus == anyag)es(e_i → honnan ∉ M^Ex) then
            ujEl(M.megtalal(m → nev == e_i → honnan → nev + "_Periodus_" +
              + per_i → nev), e_i → hova);

            P − graf.hozzaad(ujEl) ;
            P − graf.torol(e_i) ;
        end
        if (e_i → honnan → tipus == muveletiegyseg)es(e_i → honnan ∉ O^Ex) then
            ujEl(O.megtalal(o → nev == e_i → honnan → nev + "_Periodus_" +
              + per_i → nev), e_i → hova);

            P − graf.hozzaad(ujEl) ;
            P − graf.torol(e_i) ;
        end
        if (e_i → hova → tipus == anyag)es(e_i → hova ∉ M^Ex) then
            ujEl(e_i → honnan, M.megtalal(m → nev == e_i → hova → nev + "_Periodus_" +
              + per_i → nev));

            P − graf.hozzaad(ujEl) ;
            P − graf.torol(e_i) ;
        end
        if (e_i → hova → tipus == muveletiegyseg)es(e_i → hova ∉ O^Ex) then
            ujEl(e_i → honnan, O.megtalal(o → nev == e_i → hova → nev + "_Periodus_" +
              + per_i → nev));

            P − graf.hozzaad(ujEl) ;
            P − graf.torol(e_i) ;
        end
    end
end
P − graf.torol(masolandoStruktura) ;
```

Ahhoz tehát, hogy egy tárolót implementálni lehessen az alábbiakat kell tudni:

- **Tároló neve**: A modellben a beazonosítást szolgálja.
- **Beruházási költség**: Ahogyan minden műveleti egységnek, úgy a tárolónak is lehet beruházási költsége.
- **Működési költségek**: A működési költségnél felmerülhet fix költség és arányos költség is. Ez tároló esetében lehet raktár bérleti díj, vagy a kölcsönzött raklapok díja.
- **Kapacitás**: A tárolóknak a való életben mindig van kapacitásuk, azonban erre nem csak maximális, hanem minimális korlát is lehet. Ilyen az, amikor a bank előírja, hogy a számlán - ahol a pénz kerül tárolásra időről időre- kell egy minimum összegnek lennie.

– **Tárolható anyag vagy anyagok**: Meg kell határozni, hogy melyek azok az anyagok a P-gráf struktúrában, amiket a tároló képes a periódusok között mozgatni.

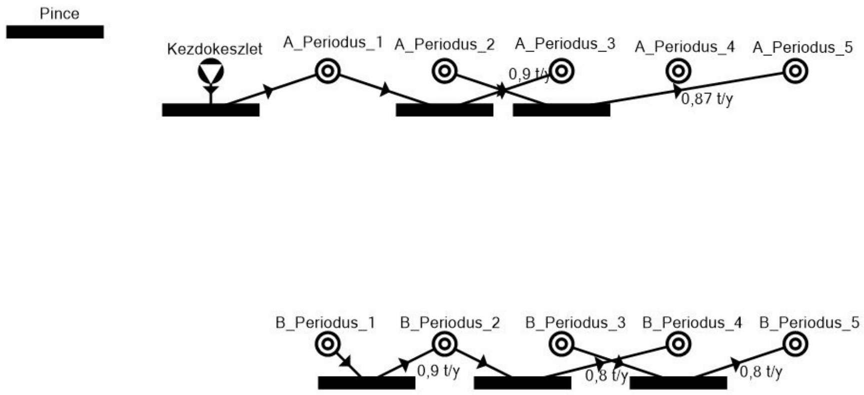
- **Kezdőkészlet**: Meg kell adni, hogy a tárolandó anyagból rendelkezésre áll-e kezdőkészlet. Ez olyan mennyiség, ami már az első periódus előtt létezik, és akár már ott is felhasználható.
- **Melyik periódusból melyikbe lehet tárolni**: Általában a tárolók minden periódus között megtalálhatók, de lehetséges olyan megkötés is, hogy a beléjük tett anyagok nem használhatók fel rögtön a következő periódusban, hanem csak az azutániban, vagy még később. Amennyiben ciklikus modellről van szó, vagyis az utolsó periódus után az első következik (például december után megint január), úgy lehetőséget kell adni arra, hogy ott is megvalósulhasson tárolás.
- **Amortizáció**: A tárolóban elhelyezett anyagok idővel veszíthetnek tulajdonságaikból, azaz amortizálódhatnak. Az amortizáció mértéke mindig ahhoz kötött, hogy mely periódusok között valósul meg a tárolás. Két egymást követő periódus között is eltérhet a tárolás amortizációja, de általában ez arányos a periódusok hosszával.
- **Kapacitáshasználat**: A tárolt anyagok bár mennyiségre megegyezhetnek, mégis más-más kapacitásra lehet szükségük. Például egy pincehelyiségben elhelyezett hordó többet vesz el a kapacitásból, mint az ugyanott elhelyezett raklap. Amennyiben több anyag kerül tárolásra, fontos tudni, hogy melyik mennyit vesz el arányosan a kapacitásból.

A következő példán keresztül szemléltetésre kerül, mennyire is összetett egy kellően általános tároló megvalósítása. Adott egy hat periódusból álló modell, ahol két anyag,"A" és "B" termékek tárolását kell megvalósítani egy pincével, aminek a beruházási költsége 500e Ft, és az alábbi tulajdonságokkal bír:

	A termék		B termék	
Kezdőkészlet	van		nincs	
Kapacitás igény	1		6	
	Mikor tárol	Amortizáció	Mikor tárol	Amortizáció
	1-3	10%	1-2	10%
	2-5	13%	2-4	20%
			3-5	20%

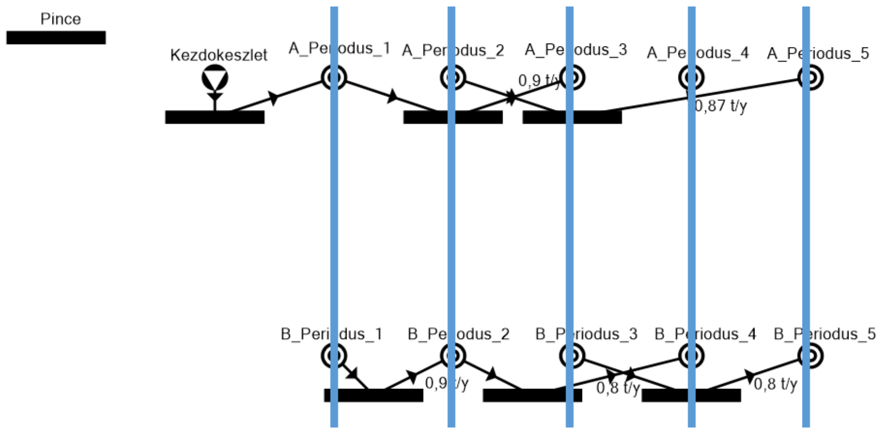
4.2. táblázat. Példaadatok tárolóhoz

A tároló implementálásánál meg kell valósítani a periódusok közötti tárolást, vagyis "A" termék esetében az 1. és 3. periódus között 10%, a 2. és 5. periódus között 13% amortizációval. Hasonlóan a három tárolási eseményt "B" termékre is létre kell hozni. A tárolások műveleti egységként jelennek meg, és az amortizáció az ezekből kifele vezető élen jelenik meg. Mivel "A" terméknek volt kezdőkészlete, ezért ennek létre kell hozni egy nyersanyagot, aminek a maximum mennyisége a kezdőkészlet, valamint egy műveleti egységet, ami az előzőekhez hasonlóan a tárolást reprezentálja. Mivel a tároló létrehozásának van beruházási költsége, ezért ez egy külön műveleti egységen jelenik meg, amivel függőségi kapcsolatban állnak majd a tárolóhasználatok. Ez az állapot látható a 4.3. ábrán.



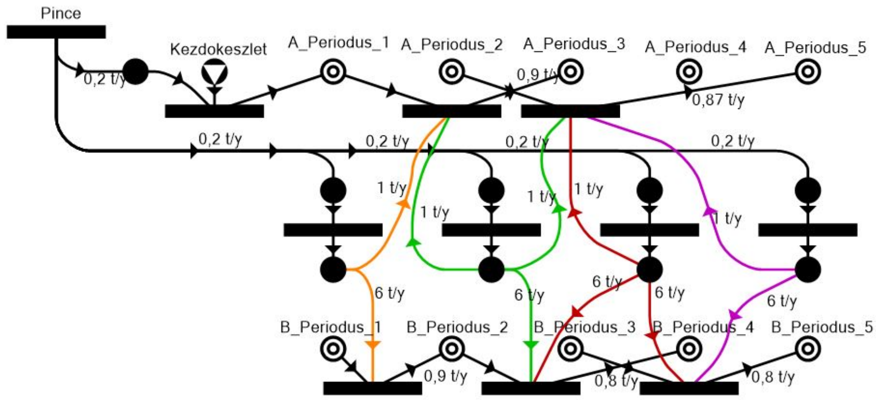
4.3. ábra. Általános tároló illusztratív példa, kapacitásleosztás előtt

Mivel a tároló képes mind a két terméket tárolni, de nem egyforma súllyal, ezért különösen figyelni kell arra, hogy egyik időpontban se léphesse túl a tárolóban tárolt elemek mennyisége a tároló kapacitását. A kereszt-tárolások miatt venni kell az összes különböző tárolási idő-metszetet, és ezekben külön-külön megvalósítani a kapacitás megkötését. A legegyszerűbben a tárolási metszeteket úgy lehet megállapítani, hogy az összes olyan helyet metszéspontnak kell jelölni, ahonnan egy tároló elkezd tárolni, vagy ahová fog tárolni. A 4.4. ábrán ezek a metszéspontok vannak bejelölve.



4.4. ábra. Általános tároló illusztratív példa, metszések helyei

Minden metszéspont között, ahol több mint egy termék érintett a tárolásban egy köztes anyag -műveleti egység- köztes anyag hármast kell felvenni. A beruházásból az első köztes anyagokba éleket kell vezetni, hiszen csak akkor használhatók a tárolók, ha megvalósult a beruházás. Itt az élek súlyát arányosítani kell a lefedett periódusok hosszával. A példában végül minden periódusközre esik ilyen elosztó, ezért 0,2-0,2 lesz az élek súlya. A műveleti egységeken fog megjelenni korlátozásként a kapacitás, vagyis egyik periódusközön sem lépheti túl a tároló ezt a határt. A műveleti egységek outputjaként megjelenő köztes anyagból a tárolásokba vezető éleken súlyként fog megjelenni az, hogy melyik termék mekkora részt vesz el a tárolóból, vagyis a kapacitás igény. Itt minden olyan tárolóba kell élt vezetni, ami érinti az adott periódusközt. Ez a súly "A" termék esetén '1', "B" termék esetén '6'. Természetesen a kezdőkészletet is be kell kötni a beruházáshoz, de mivel ott, az első periódus előtt csak egy termék érintett a tárolásban, így el lehet tekinteni az anyag-egység-anyag hármastól, és egyszerűen egy köztes anyaggal összekötni őket. A kész struktúra a 4.5. ábrán látható. A könnyebb átláthatóság végett a kapacitásszétosztás élei színessel kerültek reprezentálásra.



4.5. ábra. Általános tároló illusztratív példa, kész modell

Jól látható, hogy egy olyan általános tároló megvalósítása, ami több, különböző méretű anyagot képes tárolni nagy mértékben megnöveli a P-gráf struktúra komplexitását. A legtöbb, P-gráffal megvalósításra kerülő probléma esetében azonban elegendő egy olyan tároló, ami csak egy fajta anyagot tud tárolni. Az ilyenek jóval kevesebb hozzáadott plusz elem nélkül megvalósíthatók. Az alábbiakban bemutatásra kerül egy ilyen tároló létrehozásának pszeudokódja, ami az előző, periódusos részhez erősen kapcsolódik. A modelltranszformációt az után kell elvégezni, hogy már megtörtént a periódusokra bontás. Amennyiben a tároló csak egy anyagot képes tárolni, úgy egy tárolt anyag mindig csak egy egység terhelést jelent a kapacitásban, így kapacitáshasználatát nem kell definiálni.

A 8. algoritmusban látható, hogy az összes tárolón végigiterálva négy fő lépést kell végrehajtani. Az első a beruházás megvalósítása, a második a kezdőkészletek létrehozása, harmadik lépésben a periódusok közötti tárolást kell implementálni, majd ez után, amennyiben nyersanyag tárolásáról van szó, módosításokat szükséges eszközölni a már periódusos gráfon. Ha a tárolóhoz szükséges beruházás, úgy első lépésben az ennek megfelelő műveleti egységet kell a gráfhoz adni, hogy majd az új elemek hozzáadásakor a becsatlakoztatás egyszerűbb legyen. Ehhez egy új

Algoritmus 8: Egy-anyagos tároló megvalósítása

input : Multiperiódusos P-gráf, Anyagok eredeti nevei
output: Multiperiódusos struktúra kiegészítve tárolókkal

```
if Stor → meret! = 0 then
    foreach stor_i ∈ Stor do
        Tarolo_beruhazas ;
        Tarolo_kezdokeszlet;
        Tarolas;
        Nyersanyag_tarolasa;
    end
end
```

műveleti egységet kell felvenni *tároló neve* + "_Beruhazas" névvel, és megadni neki beruházási költségként a tároló beruházási költségét. Ennek a kódja a 10. algoritmusban látható.

Algoritmus 9: Tarolo_beruhazas

```
if stor_i.beruhazasiKoltseg > 0 then
    taroloBeruhazasMuveleti → nev = stor_i → nev + "_Beruhazas";
    taroloBeruhazasMuveleti → beruhazasi_koltseg = stor_i → beruhazasi_koltseg;
    P − graf.hozzaad(taroloBeruhazasMuveleti) ;
end
```

Második lépésként - amennyiben volt,- a kezdőkészletet kell bekötni. A kezdőkészlet olyan tárolt anyag, ami már az első periódus előtt is rendelkezésre áll, és már az első periódusban is felhasználható. Költsége általában korábban került levonásra. A kezdőkészlethez szükséges egy új nyersanyagot adni a struktúrához, aminek a maximális mennyisége a tárolónál megadott kezdőkészlet mennyiségével lesz egyenlő. Ahhoz, hogy ez összeköttetésbe kerüljön az első periódusban szereplő anyaggal, egy műveleti egységet is szükséges bevezetni, aminek a kapacitáskorlátai megegyeznek a tárolónál megadott kapacitáskorlátokkal. A két új elemet össze kell kötni egy él segítségével, valamint, az újonnan bevezetett műveleti egységet is össze kell kötni az első periódus anyagával. Ez utóbbi a neve alapján egyszerűen beazonosítható. Amennyiben beruházás útján lett létrehozva a tároló, úgy szükséges ezt, a nulladik tárolásnak is megfeleltethető műveleti egységet hozzákötni a beruházást megvalósító műveleti egységekhez. Ez egy köztes anyagon keresztül történik. Figyelni kell rá, hogy a beruházási költséget viselő műveleti egységből jövő él súlya arányosítva legyen a tárolásokon (10. algoritmus).

A beruházás és a kezdőkészlet megvalósítása után a periódusok közötti tárolást kell implementálni. Ekkor végig kell iterálni a tárolásokon. Minden tárolás az alábbi hármasból áll: kezdőperiódus (honnan), ami megmondja, hogy melyik periódusból kell tárolni, célperiódus (hova), ami azt mondja meg, hogy melyik periódusba viszi át az anyagot, valamint az amortizáció mértéke. Minden ilyen tároláshoz egy új műveleti egységet kell hozzáadni a gráfhoz. Az új műveleti egység fix költségét arányosítani kell az általa megvalósuló tárolás hosszával. (Például fix költség a havi bérleti díj, azonban, ha egy tárolás hossza két hónap, úgy ez duplázódik.) Ez

Algoritmus 10: Tarolo_kezdokeszlet

if $stor_i.kezdokeszlet > 0$ **then**

$\quad ujAnyag \to nev = stor_i \to nev + "\_" + eredetiAnyagnev(stor_i \to anyag) + \_Kezdo;$

$\quad ujAnyag \to tipus = nyersAnyag;$

$\quad ujAnyag \to maximalisMennyiseg = stor_i \to kezdokeszlet;$

$\quad P - graf.hozzaad(ujAnyag)\ ;$

$\quad ujMuveletiEgyseg \to nev = stor_i \to nev + "\_" + eredetiAnyagnev(stor_i \to anyag) +$
$\quad\quad + \_Kezdo\_muveleti;$

$\quad ujMuveletiEgyseg \to kapacitasMin = stor_i \to kapacitasMin;$

$\quad ujMuveletiEgyseg \to kapacitasMax = stor_i \to kapacitasMax;$

$\quad P - graf.hozzaad(ujMuveletiEgyseg)\ ;$

$\quad ujEl(ujAnyag, ujMuveletiEgyseg);$

$\quad P - graf.hozzaad(ujEl)\ ;$

$\quad ujEl2(ujMuveletiEgyseg, M.megtalal(m \to nev ==$
$\quad\quad = eredetiAnyagnev(stor_i \to anyag) \to nev + "\_Periodus\_" + Per[0] \to nev;$

$\quad P - graf.hozzaad(ujEl2)\ ;$

$\quad$ **if** $stor_i.beruhazasiKoltseg > 0$ **then**

$\quad\quad ujAnyag2 \to nev = stor_i \to nev + "\_" + "Beruhazasi_csatolo";$

$\quad\quad ujAnyag2 \to tipus = koztesAnyag;$

$\quad\quad P - graf.hozzaad(ujAnyag2)\ ;$

$\quad\quad ujEl3(taroloBeruhazasMuveleti, ujAnyag2);$

$\quad\quad ujEl3 \to suly = 1{,}0/stor_i \to tarolasokSzama + 1;$

$\quad\quad P - graf.hozzaad(ujEl3)\ ;$

$\quad\quad ujEl4(ujAnyag2, ujMuveletiEgyseg);$

$\quad\quad P - graf.hozzaad(ujEl4)\ ;$

$\quad$ **end**

end

az új egység továbbá átveszi a tároló arányos költségét, minimum és maximum kapacitását. Ez után a kezdőperiódusban szereplő anyagból élt kell vezetni bele, valamint egy kifele vezető él is szükséges a célperiódus anyagába. Az amortizáció miatt azonban ez utóbbi él súlya 1 - (amortizáció /100) lesz. (Az amortizáció mértéke % -os formában van megadva.) Amennyiben volt beruházási költség, ez esetben is szükséges a beruházási műveleti egységgel való összekötés. Ez egy köztes anyagon keresztül történik, illetve itt is szükséges a beruházási költség szétosztása az egyes tárolásokra, így az új műveleti egységbe vezető él súlya arányos lesz a tárolás hosszával. (11. algoritmus.)

Végül felmerülhet a kérdés, hogy mi a teendő akkor, ha nyersanyagot kell tárolni. A nyersanyag tárolásánál a probléma az, hogy ha a nyersanyag tárolóból is jöhet, akkor azt, strukturális szempontból előállítja egy műveleti egység. Ez sérti a P-gráf axiómáit, hiszen a nyersanyagot nem állítja elő semmi. Ebben az esetben a periódusoknál létre kell hozni egy-egy új nyersanyagot, ami megkapja a tárolt anyag összes tulajdonságát (a nevét is), és egy új műveleti egységen keresztül összekötni a meglévő nyersanyaggal, majd az eredeti anyag típusát megváltoztatni köztes anyagra, törölni a tulajdonságait, és a nevét *eredeti név* + "_seged"-re változtatni. Vagyis meg kell különböztetni a nem tárolható nyersanyagot, és azt, amit már lehet tárolni. Ezt a lépést végre-

Algoritmus 11: Tarolas

```
for j = 0; j < stor_i -> tarolasokSzama; j++ do
    ujMuveletiEgyseg -> nev = stor_i -> nev + "_" + j;
    P - graf.hozzaad(ujMuveletiEgyseg) ;
    periodushosszak = 0;
    for k = Per.megtalal(per == stor_i.honnan[j] + 1; k < Per.megtalal(per ==
     = stor_i.hova[j]; k++)
     do
        periodushosszak+ = Per[k] -> hossz;
    end
    ujMuveletiEgyseg -> fixkoltseg = stor_i -> fixkoltseg * periodushosszak ;
    ujMuveletiEgyseg -> aranyoskoltseg = stor_i -> aranyoskoltseg ;
    ujMuveletiEgyseg -> kapacitasMin = stor_i -> kapacitasMin ;
    ujMuveletiEgyseg -> kapacitasMax = stor_i -> kapacitasMax ;
    ujEl1(M.megtalal(m -> nev == eredetiAnyagnev(stor_i -> anyag) + "_Periodus_" +
     + stor_i.honnan[j] -> nev, ujMuveletiEgyseg);

    P - graf.hozzaad(ujEl1) ;
    ujEl2(ujMuveletiEgyseg, M.megtalal(m -> nev == eredetiAnyagnev(stor_i -> anyag) +
     + "_Periodus_" + stor_i.hova[j] -> nev, ujMuveletiEgyseg);

    ujEl2 -> suly = 1 - (stor_i.amortizacio[j]/100);
    P - graf.hozzaad(ujEl2) ;
    if stor_i.beruhazasiKoltseg > 0 then
        ujAnyag -> nev = stor_i -> nev + "_" + "Beruhazasi_csatolo" + j;
        ujAnyag -> tipus = koztesAnyag;
        P - graf.hozzaad(ujAnyag) ;
        ujEl3(taroloBeruhazasMuveleti, ujAnyag);
        if stor_i.kezdokeszlet > 0 then
            ujEl3 -> suly = 1/stor_i -> tarolasokSzama + 1;
        else
            ujEl3 -> suly = 1/stor_i -> tarolasokSzama;
        end
        P - graf.hozzaad(ujEl3) ;
        ujEl4(ujAnyag, ujMuveletiEgyseg);
        P - graf.hozzaad(ujEl4) ;
    end
end
```

hajtva már nem sérülnek az axiómák. Ez a szétszedés a 12. algoritmusban látható. A lépéseket végrehajtva a modellhez hozzáadhatók olyan tárolók, ami képesek egy-egy anyag tárolására a különböző periódusok között, különböző amortizációs mérték mellett.

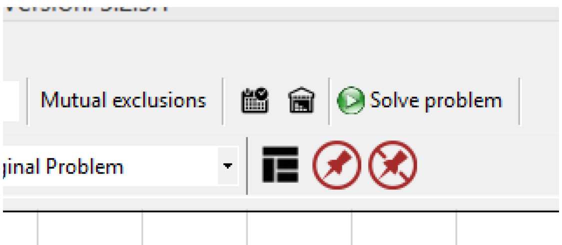
A több periódusra bontás és a tárolók hozzáadásának komplexitása $\mathcal{O}(k \times (|O| + |M|))$, ahol $|O|$ a műveleti egységek, $|M|$ az anyagok száma, k pedig a periódusszámot jelöli. Látható, hogy minél több periódusból és elemből áll egy modell, annál bonyolultabb lesz a struktúra, amin a tárolókkal történő kibővítés nem nehezít szignifikánsan.

Algoritmus 12: Nyersanyag_tarolasa

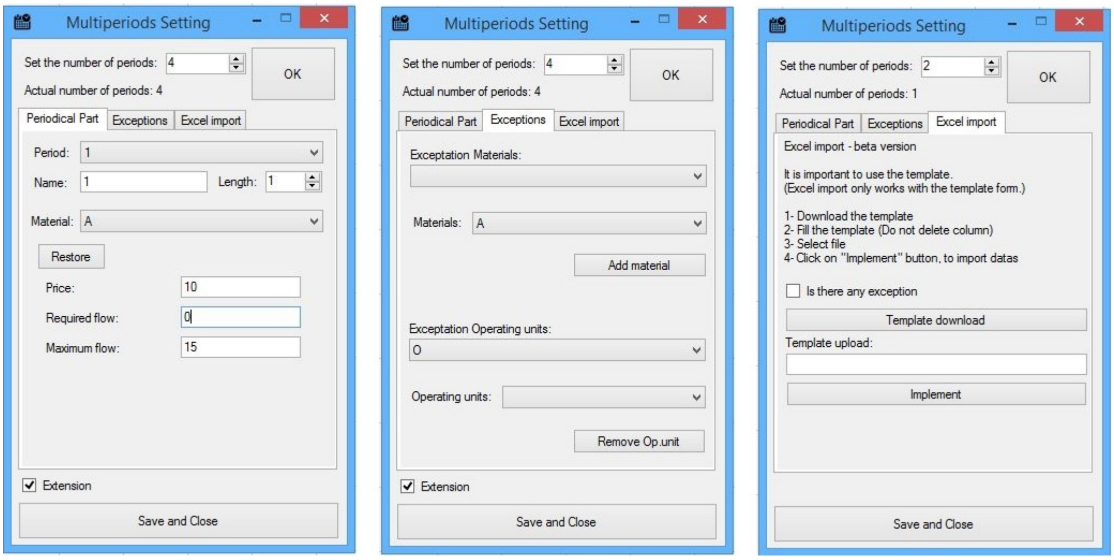
if $stor_i \to anyag \to tipus == nyersAnyag$ **then**
  foreach $per_i \in Per$ **do**
    $eredetiAnyag = M.megtalal(m \to nev == eredetiAnyagnev(stor_i \to anyag) +$
    $+ "\_Periodus\_" + per_i \to nev;$
    $ujAnyag \to tipus = nyersAnyag;$
    $ujAnyag \to ar = eredetiAnyag \to ar;$
    $ujAnyag \to minimalisMennyiseg = eredetiAnyag \to minimalisMennyiseg;$
    $ujAnyag \to maximalisMennyiseg = eredetiAnyag \to maximalisMennyiseg;$
    $ujMuveletiEgyseg \to nev = eredetiAnyag \to nev + "\_ellato\_Periodus\_" + per_i \to nev;$
    $P - graf.hozzaad(ujMuveletiEgyseg)$;
    $ujAnyag \to nev = eredetiAnyag \to nev;$
    $eredetiAnyag \to nev = eredetiAnyag \to nev + "\_seged";$
    $eredetiAnyag \to tipus = koztesAnyag;$
    $eredetiAnyag \to ar = 0;$
    $eredetiAnyag \to minimalisMennyiseg = 0;$
    $eredetiAnyag \to maximalisMennyiseg = alapertelmezett;$
    $P - graf.hozzaad(ujAnyag)$;
    $ujEl1(ujAnyag, ujMuveletiEgyseg);$
    $P - graf.hozzaad(ujEl1)$;
    $ujEl2(ujMuveletiEgyseg, eredetiAnyag);$
    $P - graf.hozzaad(ujEl2)$;
  end
end

4.4. Szoftveres megvalósítás

A P-graph Studio egy olyan keretrendszer, amely segít a P-gráffal való modellezésben és képes alkalmazni arra a 3. fejezetben már említett algoritmusokat. Mivel a periódusokra bontás nagy mértékben megnöveli az elemek számát, ezzel együtt a hibázási lehetőséget, viszont könnyen automatizálható, ezért igény merült fel arra, hogy integrálva legyen a P-graph Studio keretrendszerben egy ezt segítő rész. A periódusok megadása csak az után lehetséges, ha az alap gráf már modellezésre került. Ekkor az ikonsoron megjelenő naptár ikonra kattintva elő lehet hívni a periódusokat bekérő ablakot. Az ikonok a 4.6. ábrán, a multiperiódusos ablak és az abban elérhető fülek pedig a 4.7. ábrán láthatók.



4.6. ábra. A multiperiódusos rész ikonjai a P-graph Studioban

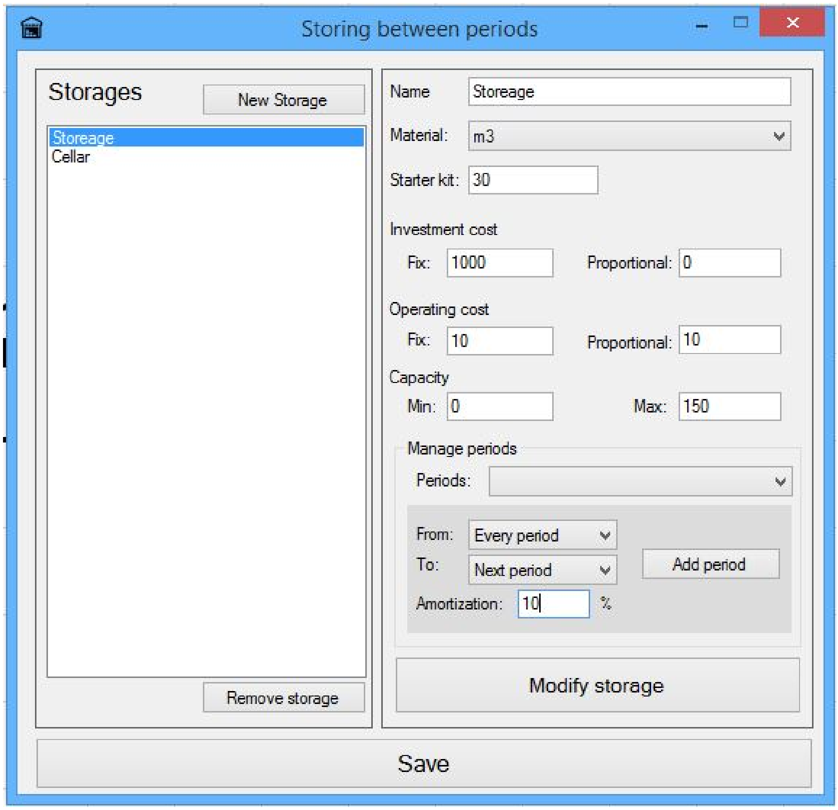


4.7. ábra. A periódusok megadására szolgáló ablak

Elsőként a periódusok számát kell megadni, majd ezt követően egy legördülő menüből lehet kiválasztani egy periódust. A periódusnak adható meg név és hossz is. Mivel ez utóbbi csak arányszámban számít, így nincs mértékegysége. Minden egyes periódusban meg lehet adni az anyagoknak külön-külön értékeket a minimum, maximum mennyiségre és az árra vonatkozóan is. Az "Extension" jelölőnégyzettel lehet megadni, hogy a periódusokkal az eredeti struktúra felbontásra vagy sokszorozásra kerül-e. Az anyagoknál, amennyiben nincs megadva érték, úgy az eredeti struktúrában megkapott értékét fogja felvenni az egyes periódusokban is.

A második fülön a kivételeket lehet megadni. Ez lehet műveleti egység vagy anyag is. Ami anyag a kivételek közé kerül az törlődik a periódusosnál kiválaszthatók közül. A harmadik fül a periódusok gyorsabb megadását segíti, és azt, amikor az anyagtulajdonságok egyenkénti megadása már időigényes lenne. Ekkor a szoftver elkészít egy sablon Excel fájlt, amit adatokkal kitöltve, és feltöltve a szoftverbe a periódusok gyorsabban megadhatók. A periódusok elmentése után a szoftver nem "szedi szét" automatikusan a struktúrát. Ez majd csak megoldáskor következik be. Az implementáció során figyelni kellett arra, hogy a program képes legyen kezelni a periódusok megadása után történő változásokat az alapstruktúrában. (Például ha periódusosnak lett jelölve valami, de végül az eredeti struktúrából a megoldás előtt ki lett törölve.)

A 4.8. ábrán már a tárolók beviteli ablaka látható. A név megadása után ki kell választani, hogy melyik anyagot lehet benne tárolni, majd, amennyiben van, megadni a kezdőkészletet. A tárolókra is vonatkoznak a műveleti egységek általános tulajdonságai, vagyis a fix és arányos költségek, a kapacitások, illetve a beruházási költségek. Az ablak jobb alsó részén lehet felvinni, hogy melyik periódusból melyik periódusba legyen képes tárolni, és milyen amortizáció mellett. Mivel leggyakrabban minden periódus között lehetséges a tárolás, így lehetőség van egyetlen felvitellel megadni azt (összes periódusból a következőbe). Megoldás után a szoftver szétbontja a periódusoknak megfelelően az eredeti struktúrát, és amennyiben volt, kiegészíti a tárolókkal is. A



4.8. ábra. A tárolók megadására szolgáló ablak

szétbontott struktúráról az eredeti struktúrára visszalépve lehetőség van rá, hogy az eredeti, nem periódusos struktúrát módosítsa a felhasználó, illetve arra is, hogy a már szétbontott struktúrával dolgozzon.

4.5. Alkalmazhatósági területek

A multiperiódusos jelleg számos helyen megjelenik a való életben. Ebben a fejezetben bemutatásra kerül néhány többperiódusú probléma, és azok megoldásai, amiből publikációk is születtek.

4.5.1. Gyártástervezés

Magyarországon az összeszerelő üzemek az ipar egy jelentős részét teszik ki, számos munkahelyet biztosítva. Az elmúlt években azonban annak ellenére, hogy ezek a munkahelyek nem igényelnek speciális szaktudást, emberhiánnyal kellett szembesülni. A gyáraknak a fokozott munkaerőhiány tekintetében még inkább figyelembe kell venni a lehetséges optimalizálási lehetőségeket. Ez már a gyártástervezés szakaszában is megvalósulhat, vagyis amikor arról kell dönteni, hogy milyen termék kerüljön következőnek a gyártósorra. Ezt több minden is befolyásolhatja:

- Megrendelések: A megrendelésekből kiderül, hogy melyik terméket kell majd legyártani, és amennyiben a rendelés határidőt is von maga után, úgy azt az időintervallumot is leszűkíti, hogy mikor lesz legyártva.

- Termékkód: A termékkódok megmondják, hogy pontosan milyen elemekből, alkatrészekből kell állnia a terméknek.

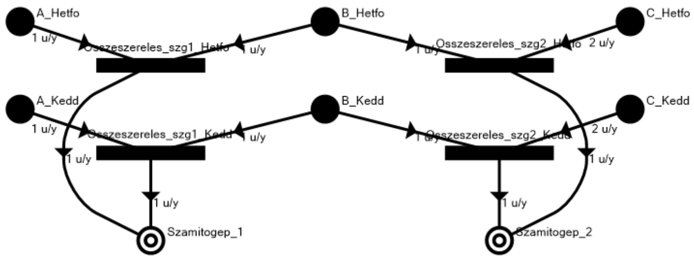
- Mennyiség: A mennyiség határozza meg, hogy mennyit kell a termékből létrehozni. Elképzelhető olyan eset, amikor az alkatrészek hiányában csak a mennyiség egy részért lehet legyártani.

- Prioritás: A magasabb prioritású megrendeléseket hamarabb kell teljesíteni. Nem egyszerű eldönteni, hogy a normál prioritású, de határidős, vagy a magasabb prioritású termék kerüljön gyártásra.

- Státusz: A gyártástervezést befolyásolhatják a termékek és alkatrészek státuszai. (Például ha egy szükséges anyag még nem érkezett be.)

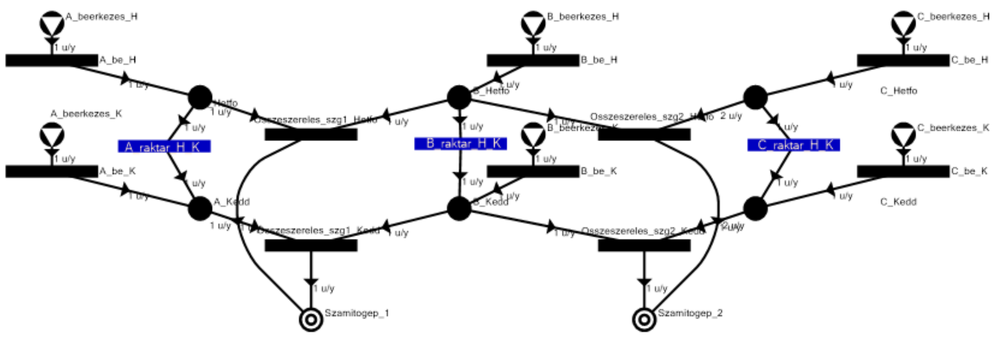
Amennyiben a gyártás már több éve folyik, úgy az időszakos trendeket is figyelembe lehet venni a tervezésnél, illetve az alkatrészek beérkezési idejét. Lehetnek úgynevezett termékcsaládok, ahol a fő részek azonosak, és csak kisebb alkatrészekben térnek el egymástól a termékek.

A cél egy olyan modell elkészítése volt, ami nem csak az aktuális állapot alapján mondja meg, hogy mely termék kerüljön következőnek gyártásba, de képes több periódussal, és ezzel együtt például az alkatrész beérkezésekkel is számolni. A gyártásnál a termékpaletta alapján adottak a termékek, illetve az, hogy mely termék melyik alkatrészekből áll, továbbá, hogy abból az alkatrészből hány darab szükséges hozzá. Ez utóbbi egy mátrixszal adható meg, ahol a sorok a termékek, az oszlopok az alkatrészek, és az érték a szükséges mennyiség. A megrendelések alapján egy-egy termék gyártásához prioritás is társulhat, ami a gyártás idejére lesz hatással. A vállalat mondja meg, hogy a magas prioritású termékeket minimum hány nap alatt kell legyártani, illetve, ha egy prioritásos terméknek a határideje egybeesik egy másik termékével, és nem megoldható a korábbi teljesítés, úgy a prioritásos termék kerül legyártásra. Adott továbbá egy mátrix, ami megadja, hogy melyik napon melyik alkatrészből mennyi fog beérkezni. Minden termékhez tartozik egy megrendelt mennyiség is. Egy alkatrész adott napi mennyisége megegyezik az előző napi mennyiséggel, kivonva belőle az előző nap felhasznált mennyiséget valamint hozzáadva az aznap beérkezett mennyiséget. Egy darab termék összeszereléséhez szükséges annak összes alkatrésze, és azokból is a szükséges mennyiség. Két vagy több különböző napi termelés adott termékre vonatkozóan összeadódik, tehát mindegy, hogy hány nap alatt, és mekkora része kerül legyártásra a termék megrendelt mennyiségének, csak a határidőig történjen meg az összeszerelés a várt mennyiségben. (Általában 28 napra kell előre tervezni.) Mivel a gyártósor egy nap csak véges mennyiségű terméket tud gyártani, így adott egy napi kapacitásmaximum is, ami az összeszerelésre vonatkozik. A cél ezek után az, hogy a termékek az alkatrészek beérkezésének figyelembevételével a lehető leghamarabb, legkevesebb átállással legyártásra kerüljenek legkésőbb a határidő napjáig, a rendelt mennyiségben.



4.9. ábra. Két termék három alkatrészből történő összeszerelése két munkanapon

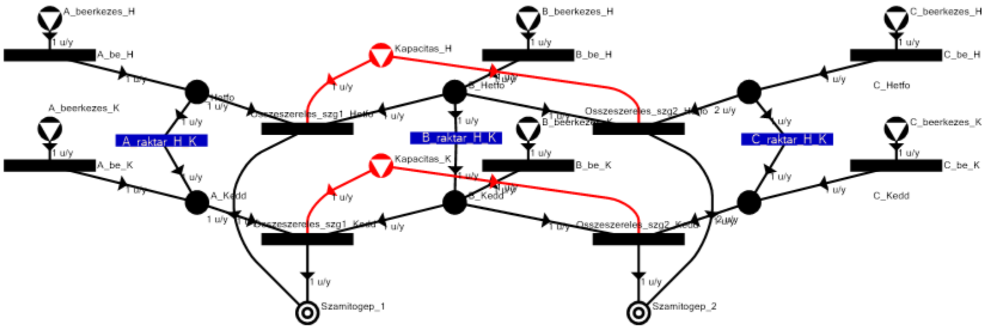
A 4.9. ábrán két termék (számítógép) összeszerelése látható három alkatrészből két munkanapon szerelve. Az ábra közepén elhelyezkedő alkatrész mindkét termékhez szükséges. A cél, hogy a termékekből a megfelelő mennyiség elkészüljön időre, de, hogy a határidő előtt mikor történik meg a szerelés lényegtelen a teljesítés szempontjából. A modell automatikusan a rendelkezésre álló alkatrészek alapján a lehető legjobb gyártási tervet fogja visszaadni. A meglévő alkatrészekből viszont az a mennyiség, ami nem kerül felhasználásra, a következő napon vagy műszakban szintén felhasználható lesz, ami tárolóval biztosítható. Az alkatrészeket folyamatosan szállítják a gyáraknak, így ezzel is számolni kell. A kiegészített modell a 4.10. ábrán látható. A kék műveleti egységek a raktárhelyiségek, amik abban segítenek, hogy a megmaradó anyagokat a következő napokon is fel lehessen használni. Modell szinten minden alkatrészhez tartozik árubeérkezés, viszont ennek értéke akár '0' is lehet. A gyártástervezők előre tudják, hogy melyik alkatrészből mikor és mennyi fog érkezni, így az a modellnek könnyen megadható.



4.10. ábra. Tárolók és alkatrész beérkezések

Számításba kell venni azt is, hogy a gyártósorok nem végtelen kapacitásúak, és naponta csak bizonyos mennyiséget tudnak előállítani egy-egy termékből. Szükséges tehát a kapacitás bevezetése a modellbe. Nyersanyagokként kerülnek reprezentálásra minden egyes műszakhoz, és az adott műszak összes összeszerelésének az egyik bemenetét adják. Ez látható a 4.11. ábrán pirossal.

A modell megalkotása mellett fontos, hogy a paraméterek a megfelelő értéket kapják meg. A



4.11. ábra. Napi kapacitás

P-gráf alapbeállításaihoz képest a következő esetekben kell eltérni: A naponként beérkező anyagnál (ami nyersanyagként jelenik meg) a maximális volumen annyi, amennyi aznap a beérkezés az egyes alkatrészekre. A termékeknél a rendelt mennyiség értéke lesz a megengedett maximális mennyiség (max.flow), továbbá a termék rendelkezik egy árral, ami az optimalizálásnál játszik szerepet. (Ez később kerül kifejtésre.) A gyártás műveleti egységénél a felső korlát a rendelt mennyiség, az alsó korlát pedig attól függ, hogy mennyire van közel a teljesítési határidő. A határidő előtti öt napon belül mindenképp le kell gyártani a termékeket. A gyártás műveleti egységén az optimalizálásnál szerepet játszó arányos költség van. A tárolt alkatrész és gyártás között az él súlya annyi, ahány abból a tárolt anyagból szükséges egy termékhez. A modell szempontjából a termék előállítása kedvező és szükséges, így egy termék létrehozása 1010$ bevételt generál a modell szempontjából. (Ez az érték nem egyenlő azzal az árral, amiért majd a vállalat értékesíteni fogja.) A gyártáson arányos költség van, és ez a költség segít végül a gyártási sorrend meghatározásában. Kiszámításához az alábbi változók szükségesek: duePrioMax (= 1), doublePrio (= 5), prioritySize (= 99), prioMinCost (= 10), prioMaxCost (= 1000), maxBoundDays (= 5). Szükségesek továbbá az alábbi, nem fix értékű változók is:

- $virtualPriority = A\ legnagyobb\ prioritás + 1 - A\ termék\ prioritása$
- $timePriority = (duePrioMax/2)(hatralevő\ napok/doublePrio)$
- *taskTimePrio = 1 + Minimum(A legnagyobb prioritás,(virtualPriority * timePriority)) / A legnagyobb prioritás * prioritySize*
- *taskValue = prioMinCost * taskTimePrio*
- *dailyValue = taskValue - (taskValue - prioMinCost) / összes nap száma * hányadik az aktuális nap*

Az gyártás arányos költsége a fenti változók ismeretében az alábbi:
$aranyoskoltseg = termekara * dailyValue.$

Az ismert adatok alapján már könnyen előállítható egy olyan P-gráf modell, ami akár több napra és több termékre is képez tervezni. Ebben segít a 13. algoritmus, aminek bemenetei az alábbiak: a *Termékek* halmaz tartalmazza a legyártandó termékeket, a *Napok* adja meg, hogy mely napokra történjen a tervezés, az *Alkatreszek* halmazban az összeszereléshez szükséges alkatrészek találhatók, az *AlkatreszBe* mátrix megadja, hogy az i-edik napon az a alkatrészből mennyi érkezik be, az *Eleme* mátrix, aminek sorai a termékek, oszlopai az alkatrészek, és a benne található értékek megadják, hogy egy termékhez az adott alkatrészből mennyi szükséges, illetve a *Kapacitas*, ami a gyártósor kapacitását jelenti.

Algoritmus 13: Gyartastervezes $\Rightarrow$ multiperiodusos PNS

input : Termekek, Napok, Alkatreszek, AlkatreszBe, Eleme, Kapacitas
output: $\mathcal{P}, \mathcal{R}, \mathcal{O}$ parameteres szintezis modell

$maxprio = 0$;
forall $t \in Termekek$ **do**
  $\mathcal{P} {:=} \mathcal{P} \cup t(cm = 1010\$, U = t \rightarrow rendeles)$;
  if $t \rightarrow priority > maxprio$ **then**
    $maxprio = t \rightarrow priority$;
  end
end
forall $i \in Napok$ **do**
  forall $a \in Alkatreszek$ **do**
    $a_j beerk_i(U = AlkatreszBe_{ai})$;
    $\mathcal{R} {:=} \mathcal{R} \cup a_j beerk_i$;
    $a_j be_i {:=} (\{a_j beerk_i\}\{a_{ji}\})$;
    $\mathcal{O} {:=} \mathcal{O} \cup a_j be_i$;
    if $i \neq |Napok|$ **then**
      $a_j raktar_i {:=} (\{a_{ji}\}\{a_{ji+1}\})$;
      $\mathcal{O} {:=} \mathcal{O} \cup a_j raktar_i$;
    end
  end
  $\mathcal{R} {:=} \mathcal{R} \cup kapacitas_i(U = Kapacitas)$;
  forall $t \in Termekek$ **do**
    $L_{szerelit} {=} 0$;
    if $Napok - i \leq 5$ **then**
      $L_{szerelit} = t \rightarrow rendeles$;
    end
    $szerel_{it} {:=} (\{kapacitas_i\}\{t\},$
     $U = t \rightarrow rendeles, L = L_{szerelit}, cp^{op} = Aranyos\_koltseg(t, |Napok|, i, maxprio))$;
    forall $a \in Alkatreszek$ **do**
      if $Eleme_{ta} > 0$ **then**
        $\alpha(szerel_{it}) := \alpha(szerel_{it}) \cup Eleme_{ta} * a_{ji}$;
      end
    end
    $\mathcal{O} {:=} \mathcal{O} \cup szerel_{it}$;
  end
end

A fentiekben bemutatott eset csupán egy reprezentatív, könnyen átlátható és megérthető példa. A valóságban a cégek több hétre előre terveznek és a termékpalettájuk is színesebb. A

Algoritmus 14: Aranyos_koltseg

input : $Termekek_t$, Napok_szama, i, maxprio
output: cost

virtual_priority=maxprio+1-$Termekek_t \rightarrow priority$;
timePriority=(1/2)((Napok_szama-i)/5);
taskTimePrio=1+$Minimum\{maxprio, (virtual\_priority * timePriority)\}/maxprio * 99$;
taskValue = $10 * taskTimePrio$;
dailyValue=$taskValue - (taskValue - 10)/Napok\_szama * i$;
cost = $1010 * dailyValue$;

fenti leírás alapján egy magyarországi vállalattal közös projekt kapcsán implementálásra került egy szoftver, ami a cég specifikus adataiból 28 napos tervezéssel létrehozza a gyártási tervet a fent ismertetett módszer segítségével. A vállalat korábban manuálisan készítette el a gyártási tervet és nem számolt a sorok kapacitáskorlátjával. Az eredmények kimutatták, hogy az optimalizálás nagyban segíti a tervezést és annak egyenletes eloszlását, hiszen a jövőben beérkező alkatrészeket is figyelembe veszi, illetve a kapacitáskorlát bevezetésével a modell sokkal jobban megközelítette a valóságot.

Nyitott kérdés maradt a munkaerő bevezetése, illetve az, hogy a modell a két termék közötti átállási időt is beleszámolja-e a tervezésbe. A munkások feladathoz rendelése nem triviális, és a gyakorlat azt mutatja, hogy nem tervezhető annyira előre. Ez az 5. fejezetben kerül részletesebben kifejtésre. A termékek közötti átállási idő minimalizálásával elérhető időnyereség a már megvalósult gyártástervezési modellhez képest szintén minimális.

4.5.2. Hulladékkezelés és karbantartási idők

A vegyiparban, és az ahhoz kapcsolódó iparágakban gyakran keletkezik a termelés során káros anyag melléktermékként, amit a hulladékgazdálkodásnak megfelelően kell kezelni. A környezeti terhelés csökkentését szerencsére már kormányzati szinten is egyre komolyabban veszik. Szükséges tehát ezt beépíteni az ellátási láncba. A káros anyagok keletkezése azonban nem egyszeri eset, hanem időről időre foglalkozni kell vele. Nehezíti a tervezést, hogy bizonyos műveleti egységek a karbantartási időszakok miatt időnként nem állnak rendelkezésre.

Szükségszerű tehát egy olyan modell kidolgozása ami képes megkülönböztetni a hulladékot, és kikényszeríteni annak valamilyen "felhasználását". Fontos, hogy több periódusra legyen képes tervezni, ahol karbantartási időszakok miatt műveleti egységek eshetnek ki. Az induló struktúra egy termelési folyamat, ami az anyagok és műveleti egységek felsorolásával egy egyszerű gyártási folyamatot ír le.

$$m_{ik} \in M, o_{jk} \in O, p_k \in P \tag{4.1}$$

ahol M az anyagok, O a műveleti egységek, P pedig a periódusok. A műveleti egységek inputjai és outputjai is anyagok.

$$o_{jk}^{+} \subseteq M, o_{jk}^{-} \subseteq M \tag{4.2}$$

További információként megjelenik, hogy mely anyag a kritikus hulladékkezelési szempontból, mivel a kritikus anyagnak mindig el kell fogynia. (Az, hogy ezt egy termelésben részt vevő műveleti egység fogyasztja, vagy direkt a hulladék eltávolítására dedikált műveleti egység, a modell szempontjából lényegtelen.)

$$m_{ik}^{c} \in M^{critical}, M^{critical} \subseteq M \tag{4.3}$$

$$\left(\sum_{i=1}^{|M|} \sum_{k=1}^{|P|} m_{ik}^{c} \right) = 0 \tag{4.4}$$

A periódusok miatt az előállítandó, vagyis a várt termék mennyiség is eltérhet a különböző időszakokban, de ettől ennél a példánál eltekintünk. Annál a periódusnál, ahol karbantartás folyik, a karbantartást érintő műveleti egységnél a termelés felső korlátja '0' lesz, egyébként a felső korlát a műveleti egység kapacitásával lesz egyenlő. A karbantartási idők megadhatók egy mátrixszal, aminek sorai a műveleti egységek, oszlopai a periódusok, és értéke '1', ha az aktuális periódusban karbantartás folyik.

$$Service_{jk} = \begin{cases} 1 & o_j \textit{ műveleti egységet } p_k \textit{ periódusban szervizelik} \\ 0 & \textit{egyébként} \end{cases} \tag{4.5}$$

$$o_{jk} \leq \begin{cases} 0 & Service_{jk} = 1 \\ capacity(o_{jk}) & \textit{egyébként.} \end{cases} \tag{4.6}$$

A periódusok között lehetséges az anyagok tárolása, de az új tároló építése beruházási költséget von maga után. A periódus végén az eltárolandó mennyiséget az alábbi képlet adja:

$$m_{ik} = m_{ik-1} + prod(m_{ik}) - cons(m_{ik}) \tag{4.7}$$

$prod(m_{ip_k})$ jelöli m_i anyag gyártását k periódusban, $cons$ pedig az anyag fogyasztását ugyanekkor. Természetesen nem lehetséges negatív mennyiséget előállítani.

$$prod(m_{ik}) \geq 0, cons(m_{ik}) \geq 0 \tag{4.8}$$

A tároló beruházása csak akkor szükséges, ha van tárolandó anyag.

$$O = O \cup \begin{cases} o^{*}{}_{m_{ik}} & \exists m_{ik} > 0 \\ \emptyset & \text{egyébként} \end{cases} \tag{4.9}$$

$$cost(o^*{}_{m_{ik}}) = cost(o^*{}_{m_{ik}}) + InvestmentCost/length(p_k) \tag{4.10}$$

$$o^+_{jk} = m_{ik}, o^-_{jk} = m_{ik+1} : o_{jk} = o^*_{m_ik}. \tag{4.11}$$

A cél, a minél nagyobb mennyiségű gyártás, melynek a termék felső korlátja szab határt, viszont a gyártásért járó pozitív érték (ami akár lehet eladási ár is) ösztönzi a termelést. Fontos, hogy ne maradjon kezeletlen kritikus anyag az utolsó periódus után. Optimálisnak tekinthető a megoldás, ha az összes kritikus anyag kezelve van, az előállított termék mennyisége pedig a lehető legtöbb a legkisebb költség mellett.

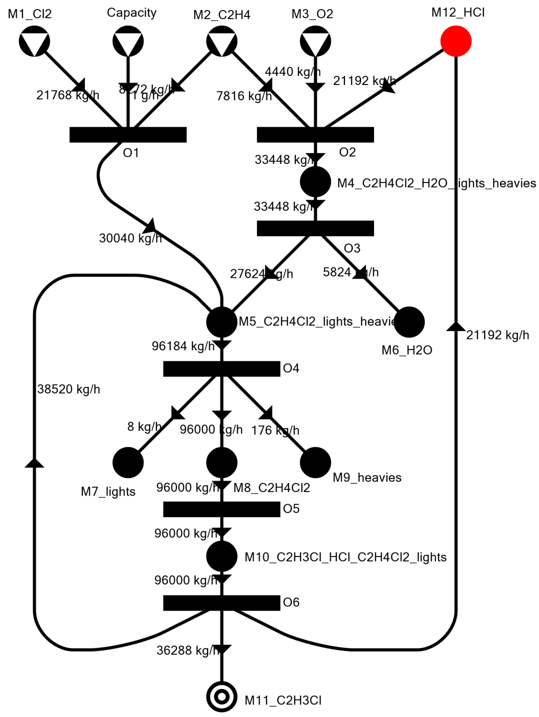
$$max \left(\sum_{k=1}^{|P|} \sum_{i=1}^{|M|} (prod(m_{ik}) * m_{ik}.value) - (cons(m_{ik}) * m_{ik}.cost)) - \sum_{k=1}^{|P|} \sum_{j=1}^{|O|} cost(o_{jk}) \right) \tag{4.12}$$

Az anyagok tekintetében a *cost* és *value* annyiban különbözik, hogy míg első a célfüggvényt negatívan érinti, és nyers, illetve köztes anyagokra vonatkozik, addig utóbbi a kész termékért kapott honorárium. Mindkettő kifejezhető pénznemben is, de értelmezhető egyfajta súlyozott értékként is. Ebben az alfejezetben egy példán keresztül bemutatásra kerül, hogyan lehet biztosítani a hulladékkezelést periódusról periódusra figyelembe véve a tárolási lehetőségek kapacitását és a karbantartási időket a folyamathálózat-szintézis segítségével.

A vinil-klorid, vagy monoklór-etilén (C_2H_3Cl) gazdaságilag az egyik legfontosabb vegyi anyag, mivel különböző polimerek, például a polivinil-klorid készül a felhasználásával [54]. Az előállítása etilénből (C_2H_4), klórból (Cl_2) és oxigénből (O_2) történik három reakciós egység segítségével. Az egyik egység az etilén (C_2H_4) direkt klórozására, a másik az oxiklórozásra, a harmadik pedig az etilén-diklorid ($C_2H_4Cl_2$) pirolízisére szolgál [55]. A hidrogénklorid (HCl), melynek vizes oldata a sósav, és ami jelen eljárásban priolízissel keletkezik, felhasználható az oxiklórozásnál, és ezáltal teljes mértékben újrahasznosítható. Természetesen minden reakcióegységhez tartozik egy vagy több szétválasztó egység, ami elkülöníti a melléktermékeket, visszaforgatja a hidrogénkloridot, vagy megtisztítja a végterméket [56].

A 4.12. ábrán a vinil-klorid előállításának modellje látható. Termékként szerepel a modellben a vinil-klorid, és nyersanyagként jelenik meg a Cl_2, a kapacitás, a C_2H_4 és az O_2. Az anyagként megjelenő kapacitás az $O1$ műveleti egység, vagyis a direkt klórozás elérhető kapacitását határozza meg. A modellben a műveleti egységeknek egyaránt van működési és beruházási költségük. A pirossal jelölt anyag a HCl, ami hulladékkezelési szempontból különös figyelmet igényel.

A modell megoldása után a HCl nem marad felhasználatlanul a struktúrában, és a maximálisan elérhető C_2H_3Cl mennyiséget lehet elérni. (A második legjobb megoldásban már jóval kevesebb ez a mennyiség, és az hulladékkezelési szempontból is elfogadhatatlan.) Amennyiben az



4.12. ábra. A vinil-klorid gyártás

$O1$ műveleti egység karbantartást igényel, úgy a kapacitás maximuma '0' lesz, ezáltal csak egy megoldás lesz elérhető: ha az egész folyamat leáll, a karbantartás idejére, annak ellenére, hogy $O2$ és $O3$ is képes lenne ellátni a modell szempontjából ugyanazt a feladatot, mint $O1$. Ezeket a leállásokat és a hulladékkezelési problémát el lehet kerülni a ciklikusság figyelembe vételével, vagyis periódusok és tárolók bevezetésével.

A hulladékkezeléshez a PNS axiómáit (3. fejezet) ki kell egészíteni és további feltételeket kell bevezetni. Ehhez meg kell különböztetni az **elvárt** és a **potenciális termékeket**. Az elvárt terméknél megkövetelt egy minimum mennyiség, amit le kell gyártani, potenciális terméknél viszont ez az alsó korlát lehet nulla. Mindkét terméktípusnak lehet ára és felső korlátja is. Egy köztes terméket mindig teljes mértében fel kell használni, ha termelésének felső korlátja nulla, vagyis nem lehet úgy gyártani, hogy a végén ne legyen felhasználva. Ezzel szemben minden olyan köztes anyag, amiből maradhat meg valamekkora mennyiség, potenciális terméknek tekinthető.

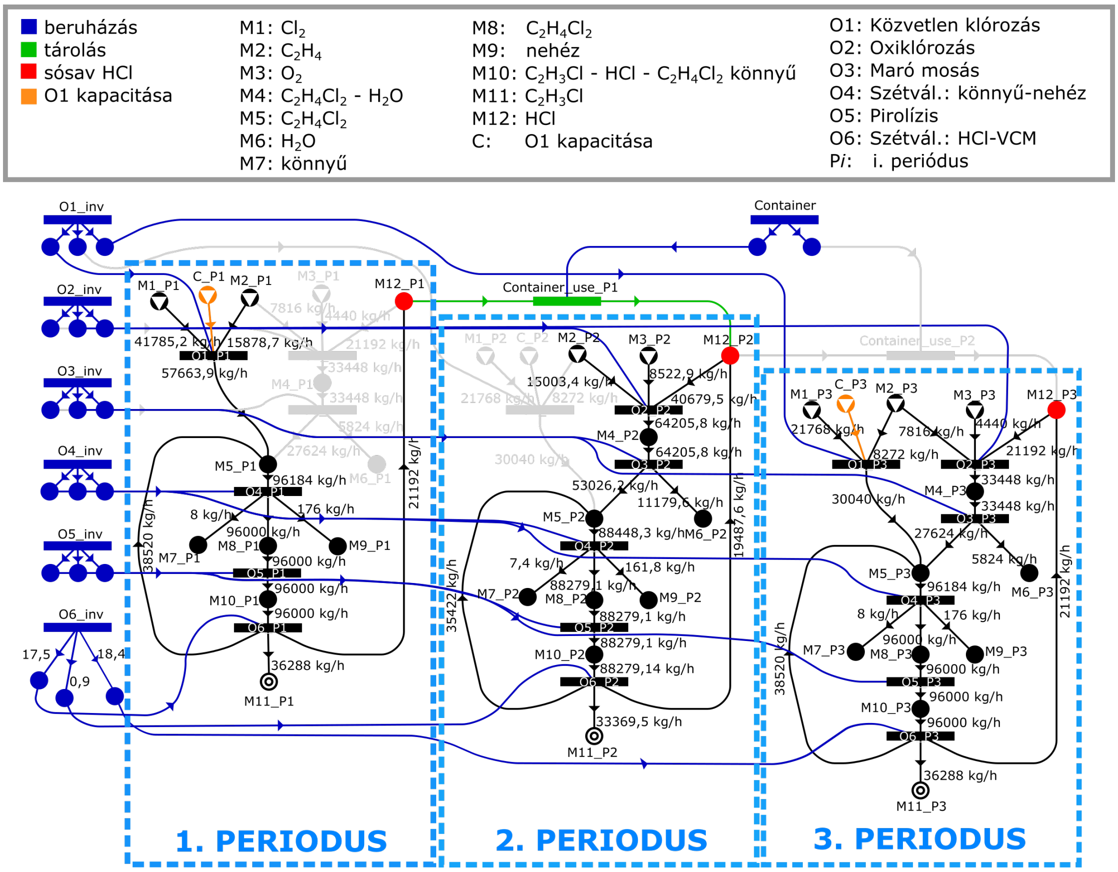
A P-gráf struktúrát meghatározó axiómákat az alábbiakkal kell kibővíteni a hulladékkezelés megfelelő modellezéséhez:

- (S1) Minden elvárt terméknek szerepelnie kell a gráfban.
- ...
- (S4) Minden műveleti egységből legalább egy út vezet elvárt- vagy potenciális termékbe.

– ...

– (S6) Minden, a folyamatok során keletkező nem kívánatos terméket legalább egy műveleti egységnek fogyasztania kell.

Az (S6)-os axióma kijelenti, hogy léteznie kell legalább egy olyan műveleti egységnek ami kezeli vagy megsemmisíti a potenciális hulladékot. A fenti példában ez teljesül is, hiszen az $O2$ műveleti egység fogyasztja a HCl-t, de a termelésének maximuma nem korlátozott. Ha ez korlátozódik nullára, úgy, bár a legjobb megoldás ugyanaz marad, de a második megoldás, ahol korábban megmaradt a HCl, eltűnik. Ha karbantartás miatt a kapacitás nulla lesz, a legjobb megoldás is eltűnik, hiszen nincs elég HCl a folyamat elindításához. Ez utóbbi eset elkerülhető egy olyan tároló telepítésével, ahol elegendő HCl raktározható el a karbantartási időszak kezdete előttről, így a folyamatok a karbantartási időszakban is elindíthatóak lesznek.



4.13. ábra. A vinil-klorid gyártás három periódusos modelljének optimális megoldása tárolókkal

Legyen tehát az alapstruktúra kiterjesztve három periódusra. Az első periódus 3800 óra, a második 200 óra és a harmadik 4000 óra hosszúságú, úgy, hogy a második periódusban karbantartással kell számolni, vagyis akkor a *Kapacitás* felső korlátja nulla lesz. Ahogy az eredeti modellben, úgy itt is a HCl -t teljes mértékben fel kell használni. Ebből a célból egy olyan tároló kerül bevezetésre, amely ezt az anyagot képes tárolni a periódusok között, ráadásul amortizáció nélkül. A tárolót fel kell építeni, így beruházási költséggel rendelkezik.

A megoldás a 4.13. ábrán látható. A piros szín továbbra is a kezelendő, veszélyes hulladékot, azaz a HCl-t jelöli. A zöld szín a tárolást, a kék a beruházást, vagy fizikai jelenlétet jelöli, míg a feketék magát a műveleti egységek használatát. Az időszakok arányos hossza a beruházást és használatot összekötő élek súlyain szerepel.

Megoldás		#1	#2	#3	#4	#5	#6
Beruházás	O1	4,04	4,04	4,04	4,04	2,11	2,11
	O2	200m	200m	200m	200m	200m	200m
	O3	200m	200m	200m	200m	200m	200m
	O4	36,78	2,11	2,11	36,78	2,11	2,11
	O5	36,78	2,11	2,11	36,78	2,11	2,11
	O6	36,78	2,11	2,11	36,78	2,11	2,11
	Tarolo	100m	1’021’190	1’021’190	1m		
1. Periódus	O1	1,92	1,92	1,92	1,92	1	1
	O2					1	1
	O3					1	1
	O4	1	1	1	1	1	1
	O5	1	1	1	1	1	1
	O6	1	1	1	1	1	1
	Tarolo	21’192	21’192	21’192	21’192		
2. Periódus	O1						
	O2	1,92			1,92		
	O3	1,92			1,92		
	O4	0,92			0,92		
	O5	0,92			0,92		
	O6	0,92			0,92		
	Tarolo		21’192	21’192			
3. Periódus	O1	1	0,08			1	
	O2	1	2	1,92		1	
	O3	1	2	1,92		1	
	O4	1	1	0,92		1	
	O5	1	1	0,92		1	
	O6	1	1	0,92		1	
Költség	USD/h	-22’923e	-16’493e	-15’930e	-15’921e	-15’120e	-8’118e

4.3. táblázat. Megoldási struktúrák

A 4.3. táblázat tartalmazza az összes lehetséges megoldást. Ahogy az az eredményekből kiderül, megéri felépíteni a tárolót, és amennyiben ez megtörténik, úgy a karbantartási időszak alatt is folytatódhat a termelés, hiszen az első periódusról marad elég HCl. Az első periódusban $O2$ és $O3$ nem vesz részt a folyamatban, mivel használniuk kéne HCl-t, viszont a második periódusban csak ők képesek biztosítani a termelést a korábbról elraktározott HCl segítségével. Az utolsó periódusban minden műveleti egység újra elérhető, így nincs szükség arra, hogy erre az időszakra HCl tárolódjon.

A második legjobb megoldás az, amikor a karbantartási időszak alatt a termelés leáll, viszont tárolás ekkor is megvalósul. Érdekes megfigyelni a bevétel drasztikus csökkenését az első és második, valamint az ötödik és hatodik megoldások között. Ez abból következik, hogy a második legjobb megoldástól egy, míg a hatodik megoldásnál már két időszak alatt is teljes leállás van. Ha nem lenne megkötés a HCl felhasználására, úgy, bár a legjobb megoldás nem változna, a megoldások száma 11-re nőne.

Egy ilyen, többperiódusú modell, ahol hulladékkezelés és karbantartási idők is szerepelnek, egyszerűen létrehozható. Ebben nyújt segítséget a 15. algoritmus, aminek bemenete az alapstruktúra, a periódusok, ha van, akkor a tárolók, a karbantartás, illetve az, hogy melyik anyag fontos hulladékkezelés szempontjából. A periódusoknál adott a hossz, a tárolóknál, azon felül, hogy milyen anyagot tárolnak, fontos tudni, hogy van-e beruházási költségük, a karbantartások mátrix soraiban pedig a periódusok, oszlopaiban a műveleti egységek szerepelnek, és "igaz" az értéke annak a cellának, ahol karbantartás folyik.

Az eredmények sikeresen bemutatják, hogy a folyamathálózat-szintézis képes arra is választ adni, hogy hogyan történjen a hulladékkezelés, valamint segíthet a karbantartási időszakokkal való tervezésben, illetve segít eldönteni különböző beruházási kérdéseket is. A megadott algoritmussal a hasonló természetű problémák könnyedén modellezhetők és reprodukálhatók.

Algoritmus 15: Hulladékkezelés és karbantartás $\Rightarrow$ multiperiódusos PNS

input : $(\mathcal{P}, \mathcal{R}, \mathcal{O})$ alapstruktúra, Periodusok, Tarolok, Karbantartas, Hulladek
output: $(\mathcal{P}^m, \mathcal{R}^m, \mathcal{O}^m)$ multiperiódusos modell

forall $o \in \mathcal{O}$ **do**
  if $cf_o^{inv} > 0$ **then**
    $o_{invest}(\{\}\{\}, cf^{inv} = cf_o^{inv})$;
    forall $i \in Periodusok$ **do**
      $\beta(o_{invest}) := \beta(o_{invest}) \cup i \rightarrow hossz * o_{invest\_i}$;
    end
    $\mathcal{O}^m := \mathcal{O}^m \cup o_{invest}$;
  end
end
forall $t \in Tarolok$ **do**
  if $t \rightarrow Investment > 0$ **then**
    $tarolo_t(\{\}\{\}, cf^{inv} = cf_t^{inv})$;
    forall $i \in Periodusok$ **do**
      if $i \neq 1$ **then**
        $\beta(tarolo_t) := \beta(tarolo_t) \cup t_{outi-1}$;
      end
    end
    $\mathcal{O}^m := \mathcal{O}^m \cup tarolo_t$;
  end
end

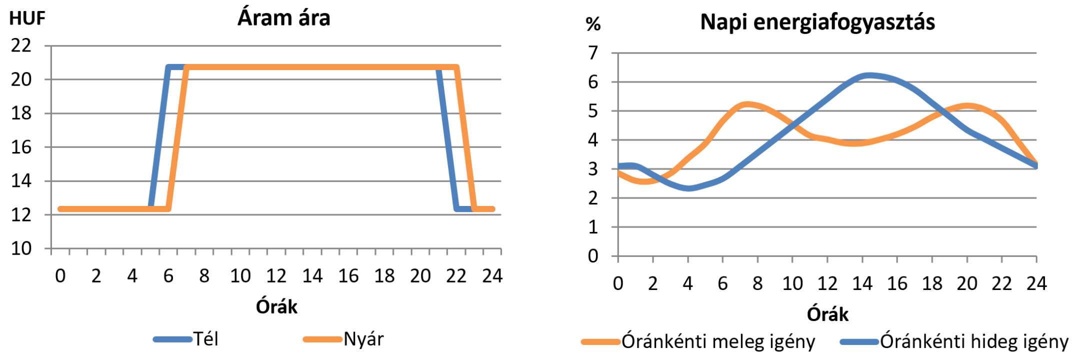
```
forall i ∈ Periodusok do
    forall p ∈ 𝒫 do
        p_i=p;
        𝒫^m:=𝒫^m ∪ p_i;
    end
    forall r ∈ ℛ do
        r_i=r;
        ℛ^m:=ℛ^m ∪ r_i;
    end
    forall o ∈ 𝒪 do
        o_i=o;
        cf_{o_i}^{inv} = 0;
        o_i({α(o) ⇒ i, o_{invest_i}}{β(o) ⇒ i});
        if Karbantartas_{io} = igaz then
            o_i(U = 0);
        end
        𝒪^m:=𝒪^m ∪ o_i;
    end
    if i ≠ 1 then
        forall t ∈ Tarolok do
            tarolo_{ti-1}({(t → anyag) ⇒ i − 1, t_{Out_i−1}}{(t → anyag) ⇒ i});
            𝒪^m:=𝒪^m ∪ tarolo_{ti-1};
        end
    end
end
forall m ∈ (α(𝒪) ∪ β(𝒪)) do
    if m ∈ Hulladek then
        m(U = L = 0);
    end
end
```

4.5.3. Energiatárolás

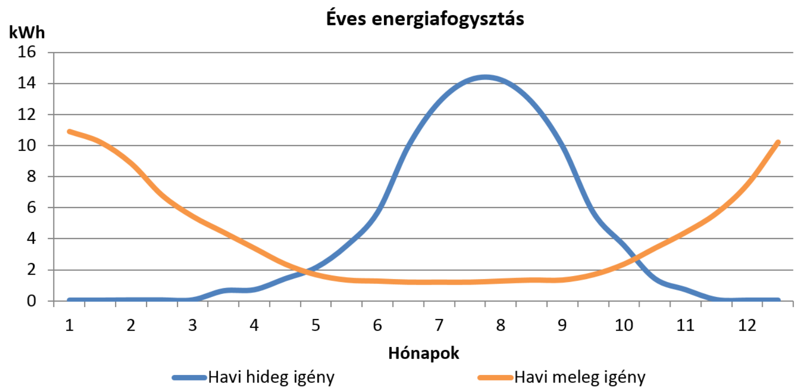
Mivel az energiahasználat jelenti a háztartások legnagyobb kiadását, vagyis a fűtés, gáz, meleg víz és a villamos energia, ezért érdemes ezek használatát valamilyen módon optimalizálni, azaz ütemezni. Egy sikeres optimalizálással azonban nem csak a háztartási költségek csökkenthetők, de a környezeti terhelés is kisebb lehet. Szerencsére ez utóbbi szempont nem csak elméleti és kutatási szinten került fókuszba, hanem a gyakorlatban is egyre nagyobb hangsúlyt kap.

Háztartási fűtési- és hűtési energia ellátás optimalizálása

A háztartások hűtési és fűtési energiaigénye a különböző periódusokban eltérő, viszont egy jól meghatározható fogyasztási profil szerint történik. Az energia ára is változhat periódusonként, ami leggyakrabban napszakhoz kötött. (Például éjszakai áram.) A 4.14. ábrán látható az áram árának alakulása egy nap, illetve a napi energiaigény. Észrevehető, hogy a fűtés és meleg víz használata reggel és este jelentősebb, ezzel ellentétben hidegre leginkább kora délután van igény (például klíma). A napi igények mellett ciklikus tendenciát mutat az éves igény is. A 4.15. ábrán láthatók a különböző igények éves lebontásban.



4.14. ábra. Áram árak és igény



4.15. ábra. Az igények éves eloszlása

A cél a háztartási hűtési és fűtési energia elosztásának optimalizálása a felmerülő költségek alapján. Ehhez adottak a meleg és hideg energiára való igények periódusokra lebontva:

$$0 \leq m_k, 0 \leq h_k \tag{4.13}$$

$$1 \leq k \leq \text{periódusszám}. \tag{4.14}$$

Adott továbbá periódusonként a gáz és áram ára:

$$0 \leq cost(g_k), 0 \leq cost(a_k) \tag{4.15}$$

Lehetséges tárolók alkalmazása, ami bizonyos veszteség mellett képes egyik periódusból a másikba átvinni az előállított hideg, vagy meleg anyagot (pl.: bojler). Ezekhez viszont beruházás szükséges, aminek anyagi vonzata van. Tároló alkalmazásakor elegendő lehet kevesebb áram vagy gáz fogyasztása, ha korábbi, kisebb költségű periódusból megmaradt a hideg vagy meleg áramból valamennyi. Minden periódusban az előállított mennyiség nagyobb, vagy egyenlő kell, hogy legyen mint a periódusra vonatkozó igény.

$$marad(m_k) = marad(m_{k-1}) * (100 - amortizacio(tarolo^m)/100) + g_k - m_k \tag{4.16}$$

$$marad(h_k) = marad(h_{k-1}) * (100 - amortizacio(tarolo^h)/100) + a_k - h_k \tag{4.17}$$

$$m_k \leq marad(m_{k-1}) * (100 - amortizacio(tarolo^m)/100) + g_k \tag{4.18}$$

$$h_k \leq marad(h_{k-1}) * (100 - amortizacio(tarolo^h)/100) + a_k \tag{4.19}$$

$$0 \leq marad(h_k), 0 \leq marad(m_k) \tag{4.20}$$

$$m^{investmentCost} = \begin{cases} investmentCost(tarolo^m) & \exists marad(m_k) \geq 0 \\ 0 & \text{egyébként} \end{cases} \tag{4.21}$$

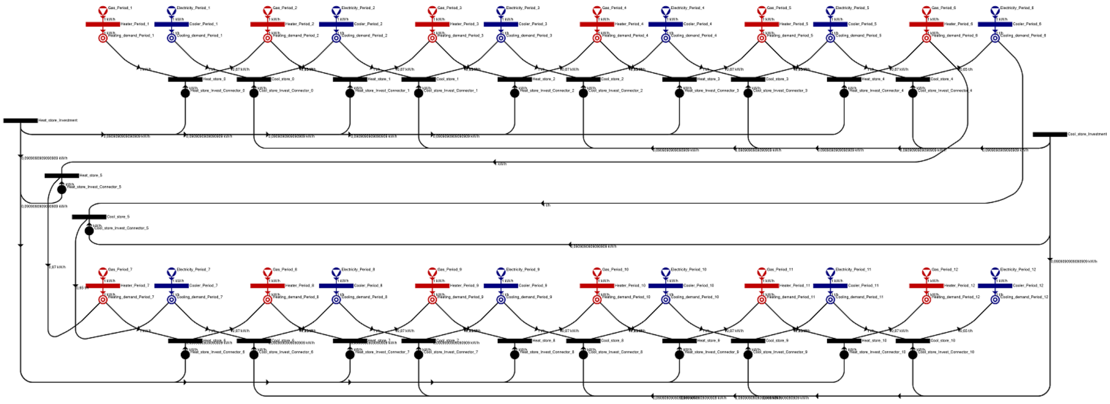
$$h^{investmentCost} = \begin{cases} investmentCost(tarolo^h) & \exists marad(h_k) \geq 0 \\ 0 & \text{egyébként} \end{cases} \tag{4.22}$$

A cél a költségek csökkentése amellett, hogy az összes igény kielégítésre kerül:

$$min\Bigg(\sum_{k=1}^{periodusszam} \Big(g_k * cost(g_k) + a_k * cost(a_k) \Big) + h^{investmentCost} + m^{investmentCost} \Bigg) \tag{4.23}$$

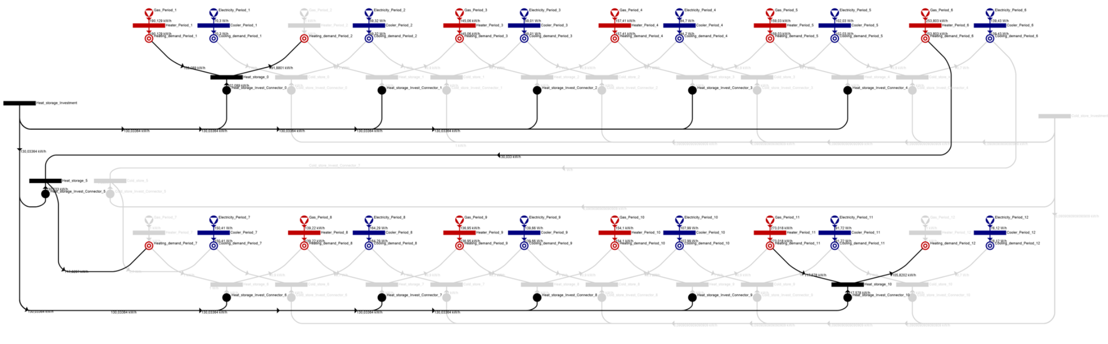
Az alap probléma P-gráf modellje viszonylag egyszerű: Egy-egy nyersanyag reprezentálja a gázt és az elektromos áramot, amikből egy-egy műveleti egység segítségével lehet a fűtést és hűtést megvalósítani. Az igények a termékeken vannak, mint elvárt mennyiség. Ebből a struktúrából kell annyit létrehozni, amennyi az igény szerinti periódusszám. Ez lehet egy heti modell fél órás bontásban, vagy akár egy éves modell napi bontásban is. A felbontás attól függ, mennyire kell részletesen és hosszan előre tervezni.

Mivel a periódusok nem függetlenek egymástól, és az előállított hideg, valamint meleg tárolására vannak eszközök, így a periódusok közötti átvitel megvalósítható. Ehhez szükséges a tárolók beruházása. A tárolás viszont mindkét esetben veszteséggel jár. A 4.16. ábrán egy ilyen modell látható, ahol egy nap igénye (január 4.) két órás felbontásban került implementálásra. A tárolás vesztesége ennél a példánál 13%.



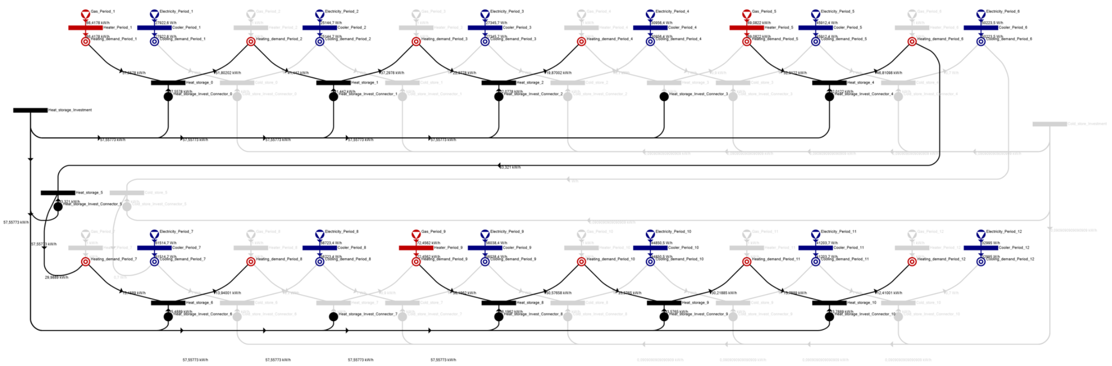
4.16. ábra. Egy 12 periódusos modell tárolással

A modellt megoldva a 4.17. ábrán látható struktúra lesz a legjobb megoldás. Látható, hogy tárolóra szükség van, viszont csak arra, ami a meleget tudja tárolni (például bojler), hiszen csak ez aktív a struktúrában. Ekkor az első, a hatodik, és a tizenegyedik periódusnál történik tárolás. Joggal merül fel a kérdés, hogy miért éri meg beruházni és tárolni ezekben az időszakokban, mintsem előállítani, amikor az előállításhoz szükséges gáz ára nem változik. A válasz abban rejlik, hogy a melegítésnek a fix költsége magas, és ezek azok az egymást követő periódusok, amikor kellően kevés az igény ahhoz, hogy a tárolóban elférjen mindkét periódus igénye. Amennyiben nem egy téli, hanem egy nyári nap kerül modellezésre, a megoldás után kiderül, hogy olyankor még inkább megéri a tároló használata (4.18. ábra). A nyári időszakban a meleg igény sokkal alacsonyabb, mint télen, ezért több órai igénynek megfelelő mennyiséget is el lehet tárolni ugyanabban a tárolóban.



4.17. ábra. A legjobb megoldás - január

Egy ilyen modell létrehozása könnyen automatizálható a szükséges információk ismeretében. A 16. algoritmus levezeti, hogyan lehet generálni egy több periódusból álló PNS modellt a háztartási meleg-, és hideg igényre. Az algoritmus bemenete a Hideg- és MelegIgény, ami minden



4.18. ábra. A legjobb megoldás - július

periódusra megmondja, hogy mennyi a háztartás elvárt mennyisége. Szükséges tudni, hogy mekkora a költsége egy egységnyi igény kielégítésének. Ez a gáz, illetve elektromos áram árával lesz arányos, és a változó árak miatt ezeket is minden periódusra meg kell adni. Szükséges továbbá tudni, hogy mennyibe kerül a tárolók telepítése, és mekkora a veszteség vagy amortizáció a tárolásnál.

Algoritmus 16: Hűtési-, fűtési energia tárolás ⇒ multiperiódusos PNS

input : HidegIgeny, MelegIgeny, GazAra, AramAra, MelegTarolo, HidegTarolo
output: ($\mathcal{P}^m$,$\mathcal{R}^m$,$\mathcal{O}^m$) multiperiódusos modell

$\mathcal{O}^m := \mathcal{O}^m \cup hidegBeruhazas(\{\}\{\}, cf^{inv} = HidegTarolo \rightarrow koltseg)$;
$\mathcal{O}^m := \mathcal{O}^m \cup melegBeruhazas(\{\}\{\}, cf^{inv} = MelegTarolo \rightarrow koltseg)$;
for $i = 1; i \leq |HidegIgeny|; i++$ **do**
    $\mathcal{P}^m := \mathcal{P}^m \cup hidegigeny_i(L = HidegIgeny[i])$;
    $\mathcal{P}^m := \mathcal{P}^m \cup melegigeny_i(L = MelegIgeny[i])$;
    $\mathcal{R}^m := \mathcal{R}^m \cup aram_i(cm = AramAra[i])$;
    $\mathcal{R}^m := \mathcal{R}^m \cup gaz_i(cm = GazAra[i])$;
    $\mathcal{O}^m := \mathcal{O}^m \cup hutes_i(\{aram_i\}\{hidegigeny_i\})$;
    $\mathcal{O}^m := \mathcal{O}^m \cup futes_i(\{gaz_i\}\{melegigeny_i\})$;
    if $i \neq 1$ **then**
        $\mathcal{O}^m := \mathcal{O}^m \cup melegTarolo_{i-1}(\{melegigeny_{i-1}, melegTaroloBeruh_{i-1}\}\{(1 - MelegTarolo \rightarrow veszteseg/100) * melegigeny_i\})$;
        $\mathcal{O}^m := \mathcal{O}^m \cup hidegTarolo_{i-1}(\{hidegigeny_{i-1}, hidegTaroloBeruh_{i-1}\}\{(1 - HidegTarolo \rightarrow veszteseg/100) * hidegigeny_i\})$;
        $\beta(hidegBeruhazas) := \beta(hidegBeruhazas) \cup hidegTaroloBeruh_{i-1}$;
        $\beta(melegBeruhazas) := \beta(melegBeruhazas) \cup melegTaroloBeruh_{i-1}$;
    end
end

Az eredmények megmutatták, hogy csak a melegített anyagot éri meg tárolni, viszont a tárolással még a beruházási költség ellenére is csökkenthető a kiadás. Felmerülhet a kérdés, hogy milyen paraméterek mellett érné meg a hideg, vagyis hűtött anyag tárolása. Erre egy megtérülésbecslés, vagy egy szenzitivitás analízis adhat választ, ami a következő bekezdésben részletezésre kerül.

Új technológiák megtérülés-becslésének módszere időszakonként változó erőforrások és igények mellett

Az új technológiák bevezetése a legtöbb esetben valamilyen beruházási költséggel jár, aminek okán a befektetőket főként az érdekli, hogy megtérül-e, és ha igen mikor, az adott beruházás. Az ilyen befektetések megtérülésének becslésében még inkább kihívást jelent, ha időszakról időszakra változnak a felhasználási igények, a rendelkezésre álló erőforrások vagy nyersanyagok, ráadásul az időszakok egymástól nem függetlenek, mert tárolásra is lehetőség van. Jogosan merül tehát fel a kérdés, hogy mik azok a paraméterhatárok, ami alatt még megéri, viszont felette már nem a tervezett beruházás. Egy megfelelően paraméterezett multiperiódusos modellt megoldva válasz kapható ezekre a kérdésekre. Az alábbiakban egy kisebb példán keresztül bemutatásra kerül, hogy hogyan lehet a P-graph Studio szoftvert használni ilyen célokra.

Egy háztartás melegítés céljából napkollektort tervez telepíteni. Az igények minden periódusra adottak:

$$0 \leq m_k, \tag{4.24}$$

$$1 \leq k \leq \text{periódusszám}. \tag{4.25}$$

A megtérülés kiszámítását nagyban nehezíti, hogy nem egyforma mértékben képes előállítani a meleg vizet, hiszen a termelése függ az időjárási viszonyoktól: napos időben többet képes termelni, borús időben kevesebbet, éjszakai termelése pedig elhanyagolhatóan kevés. Ebből kifolyólag szükséges minden periódusra egy olyan előrejelzés, hogy a napkollektor a maximális kapacitásának hány százalékát tudja előállítani.

$$0 \leq n_k, \tag{4.26}$$

A kollektorral k periódusban előállítható meleg víz erősen függ a kapacitástól:

$$n_k * kollektor^{kapacitas}. \tag{4.27}$$

A napkollektor mellett a telepítés után is megmarad az a lehetőség, hogy amennyiben nem elég a kollektorral előállított meleg víz, úgy azt a szolgáltatótól vett gáz segítségével elő lehet állítani, de ez plusz költségekkel jár. Egy periódusban egységnyi meleg vízhez felhasznált gáz mennyisége az alábbi:

$$0 \leq g_k \tag{4.28}$$

A felmelegített vizet a periódusok között, vagyis a nap folyamán, és a napok között bojlerben lehet tárolni, ami már rendelkezésre áll, viszont tárolás közben a víz hőt veszthet, így amortizációval kell számolni. A periódus végén megmaradó meleg víz kiszámítható az alábbiak szerint:

$$marad(m_k) = marad(m_{k-1}) * (100 - amortizacio(tarolo^m)/100) + g_k + (n_k * kollektor^{kapacitas}) - m_k. \quad (4.29)$$

A szükséges igényt mindenképpen elő kell állítani:

$$m_k \leq marad(m_{k-1}) * (100 - amortizacio(tarolo^m)/100) + g_k + (n_k * kollektor^{kapacitas}). \quad (4.30)$$

A kollektort nem csak fel kell építeni, de annak beruházási költsége is van, így ráfordítást igényel. Továbbá figyelembe kell venni, hogy mekkora kapacitású napkollektor az, amivel a várt mennyiség elérhető. A kollektor, mivel meg kell építeni, rendelkezik beruházási költséggel és ebből kifolyólag kifizetési periódussal is, ami a megtérülésnél számít. Ez utóbbi egy ilyen beruházás esetén több éves időtartam, így a beruházási költség arányosan fog megjelenni.

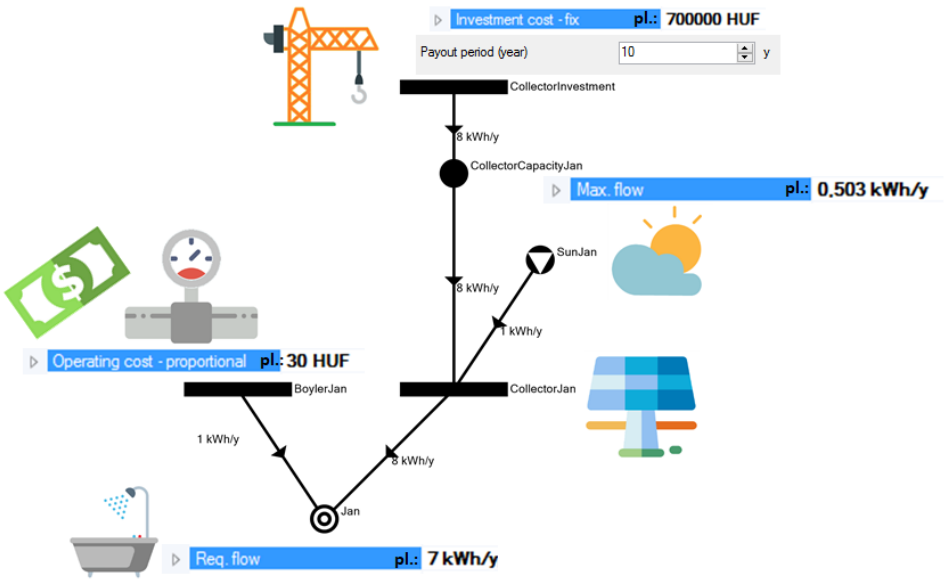
$$n^{investmentCost} = \begin{cases} investmentCost(n) & \exists n_k * kollektor^{kapacitas} \geq 0 \\ 0 & \text{egyébként} \end{cases} \quad (4.31)$$

A szolgáltatótól vett gáz alapján kiszámítható, hogy egységnyi meleg vizet milyen költségen lehet előállítani, illetve meghatározható, hogy a periódusok között mennyit hűl a tárolt meleg víz. A tárolásnak jelenleg nincsenek költségei. A modell mértékegységei mindig egységnyi meleg víz előállításához viszonyulnak. A cél a meleg víz előállítás költségének csökkentése:

$$min\Big(\sum_{k=1}^{periodusszam} \big(g_k * ar(g)\big) + n^{investmentCost}\Big). \quad (4.32)$$

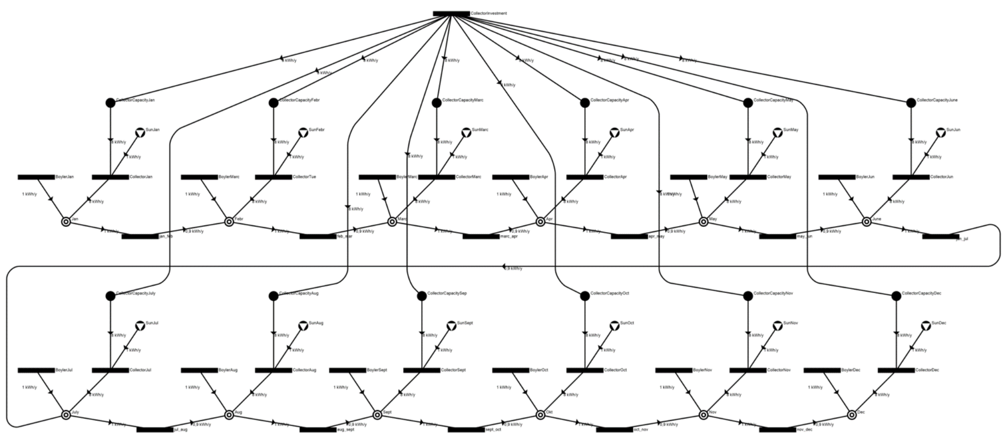
A probléma egy napi modellje a 4.19. ábrán kerül szemléltetésre. Látható, hogy a gázszolgáltatás műveleti egységként jelenik meg, hiszen mindig elérhető. Ahhoz, hogy kollektorral is lehessen meleg vizet előállítani, szükséges egyfajta beruházás, illetve a használatához arra az információra is szükség van, hogy mennyi a beérkező napsugárzás. Ez nyersanyagként van bekötve a kollektor használatához. A beruházást és a beérkező napsugárzást kétféleképp is meg lehet valósítani egy P-gráfban. Az egyik megközelítés, hogy a kollektor kapacitáskorlátja a műveleti egységre kerül, a beérkező sugárzásnál pedig pontos érték van megadva felső korlátnak. Az itt bemutatott modellben azonban a kapacitás az éleken jelenik meg, így a beérkező sugárzás mértéke egy arány lehet: értéke '1', ha teljes kapacitáson tud üzemelni, '1/2', ha csak a kapacitásának felét tudja kitermelni, ...stb.

A probléma éves modellje havi bontásban a 4.20. ábrán látható. Mivel itt már több periódus is szerepel a modellben, megjelenik a bojler, mint egy fajta tároló a periódusok között, ahol a víz hűlése, amortizációként a periódusokba érkező éleken kerül megjelenítésre. Amennyiben a modell megfelelően került felparaméterezésre, azt megoldva egyszerűen látható, hogy megéri-e



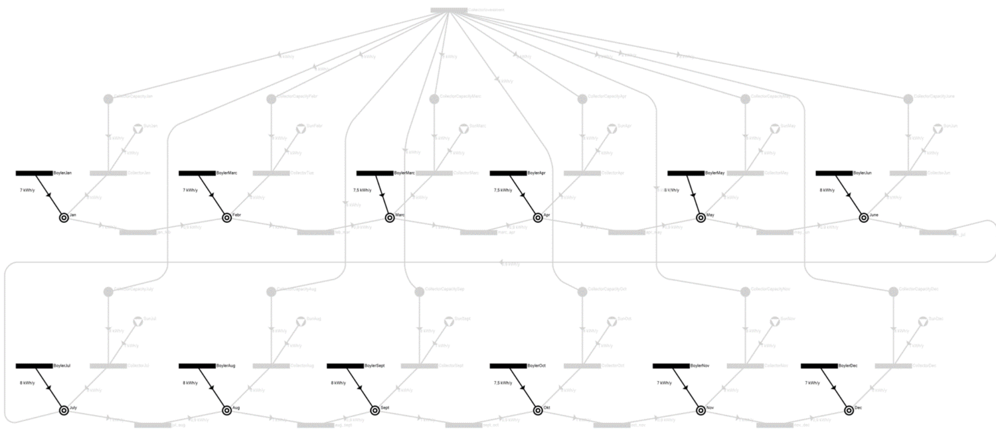
4.19. ábra. Egy napi modell

az adott beruházás, vagy sem. A beruházás akkor éri meg, ha a megoldásban az ábrázolt modell felső részén található beruházást megtestesítő műveleti egység aktív. Ez olyankor is aktív lehet, amikor szükséges gázt is venni a melegítéshez.



4.20. ábra. Éves modell havi bontásban

Abban az esetben, ha a beruházás nem térül meg, a megoldásstruktúrában inaktív marad a beruházási műveleti egység. Ez látható a 4.21 ábrán is. Ebben az esetben inaktív minden kollektorhoz köthető anyag és műveleti egység, valamint a tárolás sem éri meg, hiszen minden nap ugyanolyan költségen lehet gázt venni a szolgáltatótól és azzal előállítani a meleg vizet, amely esetben a víz hűlésével, vagyis amortizációval sem kell számolni. Felmerülhet a kérdés, hogy mik



4.21. ábra. Nem térül meg a beruházás

azok a határok, amiken belül még mindig megéri a kollektor beruházása, és mikor nem éri már meg. Ennél a példánál ez leginkább négy fő paraméterrel befolyásolható:

- Tervezett megtérülési időszak
- Kollektor beruházási költsége
- Kollektor mérete
- Szolgáltatótól vett gáz ára

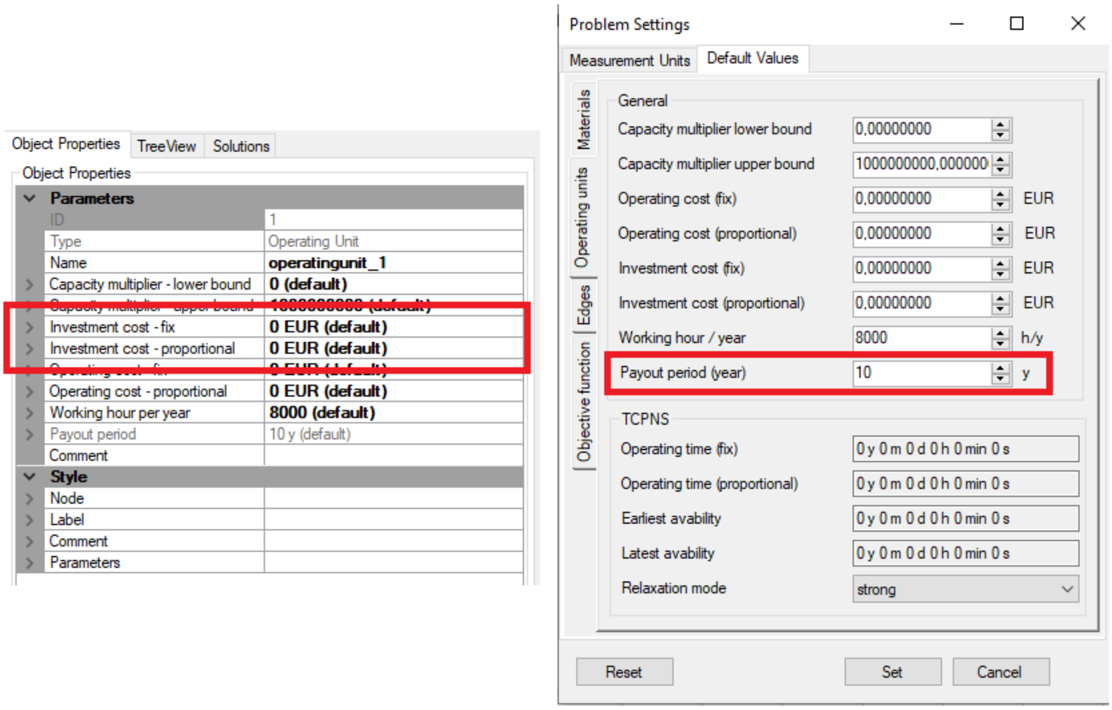
Ezek a paraméterhatárok közelítéses módszerekkel jól beazonosíthatók, változtatásukkal és a modell újbóli megoldásával pedig megállapítható, hogy az új érték mellett megéri-e az új befektetés, vagy sem. Egy beruházás gondolata sok esetben a beruházási költség miatt kerül elvetésre. A kollektor telepítésénél is felmerülhet, hogy melyik az az összeg, amit még érdemes ráfordítani, és mennyi fölött nem fog már megtérülni. A beruházási költséget a műveleti egység viseli. Ugyanígy meg lehet vizsgálni, hogy kisebb kollektorral kevésbé vagy jobban érné-e meg. Ez esetben a kollektor kapacitásmaximumán kell változtatni. A szolgáltatótól vett gáz ára is erősen befolyásolhatja a megtérülésszámítás eredményét, ahogyan a tervezett megtérülési időszakok hossza is hatással van rá.

Egy ilyen beruházás lemodellezésében segít a 17. algoritmus, ahol a bemeneti adatok sorban a következők: gáz ára (amivel egységnyi meleg vizet lehet előállítani), meleg víz igény minden periódusra, időjárás-előrejelzés a kollektor termeléséhez, a kollektor tulajdonságai, úgy, mint a beruházási költség vagy a kapacitás, valamint annak a mértéke, hogy a tárolt meleg víz mennyire hűl le két periódus között. A P-graph Studioban történő modellezésnél a szoftver lehetőséget nyújt a műveleti egységek beruházásának megadására (4.22. ábra, bal oldal), illetve a tervezett kifizetési időszak hosszának beállítására is. Ezt a programon belül a "Preferences": "Problem Settings" menüben, azon belül is az "Operating units" fül alatt adható meg (4.22. ábra, jobb oldal). A modellt megoldva egyszerűen megkapható, hogy az adott beruházásba megéri-e invesztálni.

Algoritmus 17: Meleg víz igény és kollektorberuházás ⇒ multiperiódusos PNS

input : GazAr, MelegvizIgeny, Elorejelzes, Kollektor, Amortizacio
output: ($\mathcal{P}^m$,$\mathcal{R}^m$,$\mathcal{O}^m$) multiperiódusos modell

$\mathcal{O}^m := \mathcal{O}^m \cup kollektorBeruhazas(\{\}\{\}, cf^{inv} =$
$= Kollektor \rightarrow beruhazasiKoltseg/Kollektor \rightarrow kifizetesiIdoszak)$;
for $i = 1; i \leq |MelegvizIgeny|; i++$ **do**
  $\mathcal{P}^m := \mathcal{P}^m \cup melegvizIgeny_i(L = MelegvizIgeny[i])$;
  $\mathcal{R}^m := \mathcal{R}^m \cup termeles_i(U = Elorejelzes[i])$;
  $\mathcal{O}^m := \mathcal{O}^m \cup gazzalMelegites_i(\{\}\{melegvizIgeny_i\}, cp^{op} = GazAr)$;
  $\mathcal{O}^m := \mathcal{O}^m \cup kollektorralMelegites_i(\{termeles_i, Kollektor \rightarrow kapacitas *$
  $* beruhazas_i\}\{Kollektor \rightarrow kapacitas * melegvizIgeny_i\})$;
  $\beta(kollektorBeruhazas) := \beta(kollektorBeruhazas) \cup Kollektor \rightarrow kapacitas * beruhazas_i$;
  if $i \neq 1$ **then**
    $\mathcal{O}^m := \mathcal{O}^m \cup tarolas_{i-1}(\{melegvizIgeny_{i-1}\}\{Amortizacio * melegvizIgeny_i\})$;
  end
end

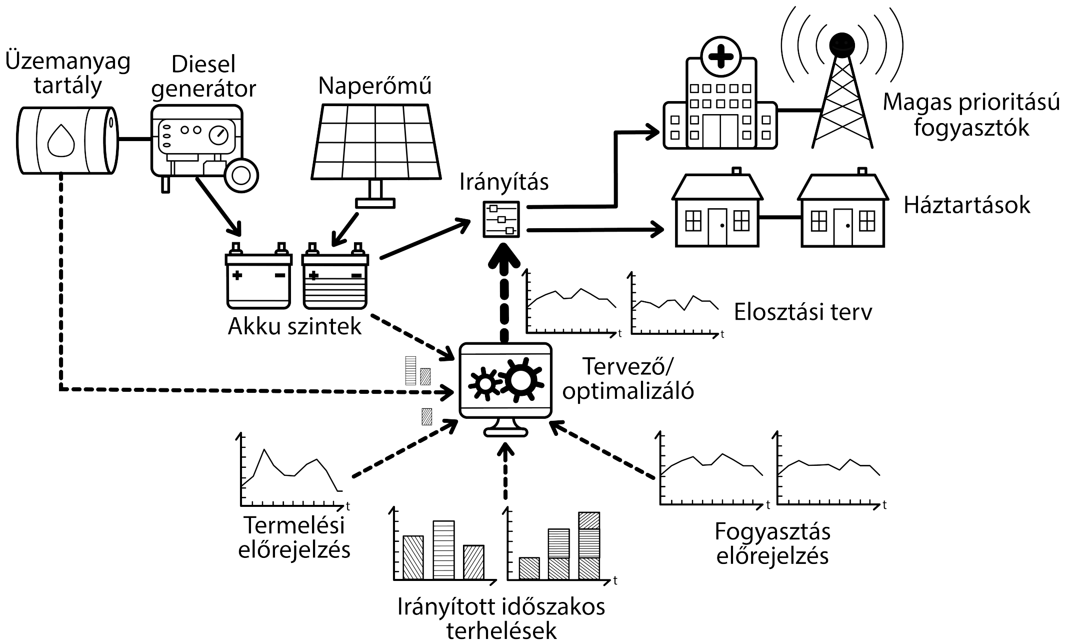


4.22. ábra. Beruházási költség és kifizetési időszak beállítása a P-graph Studioban

Megújuló forrásból származó energia tárolása és elosztása microgrid rendszerben

Az okos város, SmartCity koncepciónál a döntéshozók gyakran használják a rendelkezésükre álló információkat arra, hogy az árakon keresztül befolyásolják a fogyasztást a villamosenergia-hálózatokban, ahol az energiát nem csak megújuló forrásból tudják biztosítani. Számos esettel lehet találkozni, ahol a fogyasztói szokások megfigyelése alapján változtatják az energiaárakat

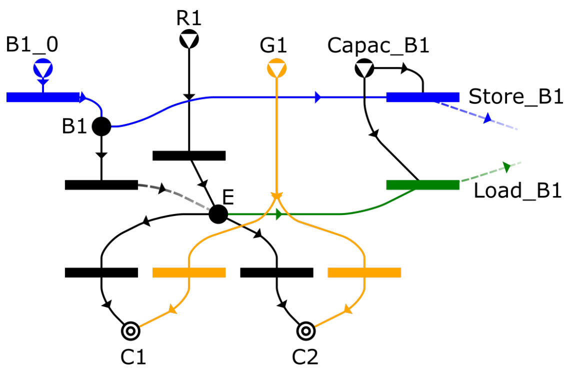
[57]. Az ilyen hálózatok, ahol nagy mennyiségű adat áll rendelkezésre a fogyasztói szokásokról, nem csak az elektromos hálózatok esetében használhatóak ki, de akár hulladékkezelésnél is jól hasznosíthatóak [58]. A többperiódusos forgatókönyvek széles körben jelen vannak az ellátási láncok különböző részein, de ez még nem jellemző az energiaellátási területeken. Napjainkban a Pinch Analízis vált népszerűvé az elektromos hálózatok optimalizálásánál, de még mindig nincs egy olyan módszer, amely hosszabb időtávval kalkulálna az energiaelosztásról, és ami a korábbi előrejelzéseket venné alapul [59]. A villamosenergia-hálózatok korszerűsítésének köszönhetően az intelligens hálózatokat nemzeti szinten is üzemeltetik, és bár van valamilyen szintű fogyasztói megkülönböztetés, ez általában a magas és az alacsony feszültség alapján történik és nem pedig prioritás alapú [60].



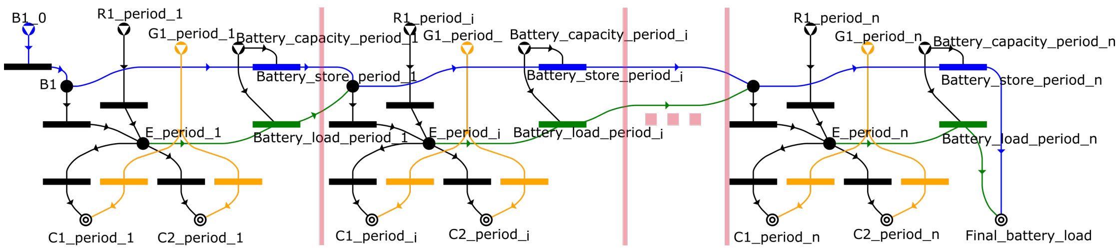
4.23. ábra. Az áramellátás alapstruktúrája

Az alábbiakban egy olyan optimalizációs módszer kerül bemutatásra, ahol az elektromos hálózatban főként megújuló forrásból származik az energia, viszont emiatt nem folyamatosan biztosított, illetve a fogyasztók prioritás alapján is megkülönböztetésre kerülnek. A megújuló forrásokon kívül nagyon indokolt esetben használható egy dízelgenerátor, de ennek használata a magas költségek miatt elkerülendő. A hálózat tartalmaz továbbá akkumulátorokat korlátozott kapacitással, amik segíthetnek a periódusok közötti energiaelosztás kiegyensúlyozásában. Az optimalizálási cél a fogyasztóknak juttatott elektromos energia mennyiségének maximalizálása, de legfejlebb annyi, amennyi azok igénye.

Az energiaellátás tervezése véges időhorizonton történik, például egy napra, vagy egy hétre kalkulálva. Ezt az időszakot több kisebb periódusra kell bontani annak megfelelően, hogy mennyire részletes adatok állnak rendelkezésre a megújuló energiaforrásokhoz és a fogyasztáshoz. Ezek az adatok már fél-, vagy negyed órás pontossággal is rendelkezésre állnak. A modell központjában az energiamennyiség áll, amit a periódusok alatt mind termelnek, mind pedig fogyasztanak az egyes entitások. Az áramszolgáltatás különböző prioritású fogyasztók között valósul meg. A hálózat továbbá tartalmazhat akkumulátorokat, amik képesek a töltés tárolására, viszont különböző állapotuk miatt (új, vagy használt) különböző hatásfokkal képesek ezt tenni. Az akkumulátoroknak lehet kezdeti töltöttsége, amit már az első periódus alkalmával is fel lehet használni, illetve lehet igény arra, hogy az utolsó számított periódus után mennyi töltöttség maradjon benne. Annak érdekében, hogy a kritikus, vagyis magasabb prioritású fogyasztó minden időszak alatt biztosan megkaphassa a szükséges áramellátást, a modell tartalmaz költségesebb, de alternatív megoldást, például egy dízelgenerátort. A korlátozott kapacitás miatt ebből is lehet több a modellben, ahogyan a megújuló erőforrást használó termelőből is. A 4.23. ábra a rendszer architektúráját mutatja be, beleértve az energiaelosztási vezérlést és az optimalizálást. Az ábrán látható, hogy az optimalizáló információt kap az üzemanyagtartályról és az akkumulátor töltöttségi szintjeiről, valamint az energiatermelésre és -felhasználásra vonatkozó előrejelzésekről az elkövetkező periódusokra vonatkozóan. A modell a rendelkezésére álló adatok alapján egy áramelosztási tervet készít.



4.24. ábra. Egy periódus struktúrája



4.25. ábra. Többperiódusos modell

A 4.24. ábrán látható a modell egyperiódusú modellje, ahol a főbb egységek kiemelésre kerültek. Legyen

$$C, E, R, B, G \in M, \tag{4.33}$$

ahol C a fogyasztók halmaza, E az energiamennyiség, R a megújuló energiaforrást használó termelők, B az akkumulátorok halmaza, G pedig azon termelők halmaza, amik költségesen, de mindig tudnak áramot biztosítani. A probléma többperiódusos modellje a 4.25. ábrán látható. Itt már jól megfigyelhető a periódusok kapcsolódása egymáshoz a tárolt energián keresztül.

A felhasználható áram maximális mennyiségét i-edik periódusnál az alábbi képlet adja:

$$E_i = MaxFlow(R_i + B_i * (100 - UsageAmort(B))), \tag{4.34}$$

$$1 \leq i \leq N, \tag{4.35}$$

ahol N a periódusok száma és

$$R_i = \sum_{j=1}^{|R|} MaxFlow(R_i^j), \tag{4.36}$$

$$B_i = \sum_{j=1}^{|B|} MaxFlow(B_i^j), \tag{4.37}$$

$$UsageAmort(B) = \sum_{j=1}^{|B|} UsageAmort(B^j), \tag{4.38}$$

ahol $|B|$ a rendszerben lévő akkumulátorok, $|R|$ pedig a megújuló erőforrást használó termelők száma. (Ha két akkumulátor van a modellben, akkor $|B| = 2$.) Mivel egy akkumulátorból nem lehet tökéletesen 100%-ban visszakapni a benne lévő töltöttséget, így egy minimális veszteséggel számolni kell. R_i^j jelöli az i-edik periódusban a j-edik, megújuló forrást használó termelő által megtermelt energiát, B_i^j pedig az i-edik periódusban a j-edik akkumulátor töltöttségét. R_i^j értékét a rendelkezésre álló adatok adják, mint például időjárás-előrejelzés napelem esetén. (Ennek a kiszámítása, vagyis, hogy milyen időjárás mekkora energiával jár, már túlmutat a dolgozat keretein. A megvalósult modellben ezek készen kapott, kiszámított adatok.) Az akkumulátor töltöttségét az alábbiaknak megfelelően lehet kiszámítani:

$$B_i^j = \begin{cases} B_{init}^j & i = 1 \\ Store\_B_{i-1}^j * (100 - StorageAmort(B^j)) + & \\ Load\_B_{i-1}^j * (100 - LoadAmort(B^j)) & egyébként, \end{cases} \tag{4.39}$$

ahol

$$Store\_B_i^j + Load\_B_i^j \leq Capac\_B^j \tag{4.40}$$

$$Store\_B_i^j = B_i^j - Usage(B_i^j) \tag{4.41}$$

$$Load\_B_i^j = E_i - (C_i - G_i). \tag{4.42}$$

$$Load\_B_i^j + Store\_B_i^j \geq Remain(B^j); \quad i = |N|. \tag{4.43}$$

B_{init}^j jelöli a j-edik akkumulátor kezdeti töltöttségét. A korábban említett amortizációhoz hasonló veszteség keletkezhet az akkumulátorok tárolásánál ($StorageAmort$), valamint töltésénél ($LoadAmort$). $Store\_B_i^j$ jelöli a j-edik akkumulátor i-edik periódus utáni tárolt mennyiségét, $Store\_B_i^j$ pedig az i-edik periódusban való töltését. $Usage(B_i^j)$ azt jelöli, hogy a j-edik akkumulátorból az i-edik periódusban mennyit használtak a fogyasztók. Mivel az akkumulátoroknak van egy fix kapacitásuk, ami fölé nem mehetnek, ezért biztosítani kell, hogy a tárolás és a visszatöltés összege ezt ne haladhassa meg. Erre hivatott a $Capac\_B^j$, mely a modellben minden periódusnál külön-külön meg fog jelenni, de értéke állandó. G_i az i-edik periódusban a nem megújuló energiaforrást használó termelőknek jelzi a használatát, vagyis amennyi áramot azok szolgáltattak a fogyasztóknak.

A modell értékét az növeli, ha minél több fogyasztói igény kerül kielégítésre, ezért szükséges az igények kielégítésére valamilyen "honoráriumot" alkalmazni. A modellben bár ez bevételként fog megjelenni, nem valódi bevételről van szó, csupán a modell befolyásolására szolgáló összegről, így a továbbiakban "értékként" lesz jelölve. Ezen értékeken keresztül lehet közvetve megadni a prioritást. A magasabb prioritású fogyasztónak magasabb, míg a kisebb prioritású fogyasztónak alacsonyabb lesz ez az értéke, ezért a modell elsőként mindenképp a magasabb prioritású fogyasztókat fogja kiszolgálni. A generátorra hasonló módon valamilyen "költséget", negatív értéket kell állítani, hiszen használata elkerülendő. A feladat meghatározásától és a modell beállításától függ, hogy melyik fogyasztó honnan kaphat áramot. Ezt modell szinten az értékeken keresztül lehet befolyásolni. Íme, néhány példa kétfogyasztós és egy generátoros esetben:

- $Value(C_1) \geq Value(C_2) \geq Cost(G)$: Ebben az esetben mindként fogyasztó használhatja a generátort. Mivel az "költséggel" jár, így a modell továbbra sem azt fogja elsődlegesnek tekinteni, és csak szükség esetén fogja használni. Ez azzal is jár, hogy minden periódusban minden fogyasztó teljes mértékben kielégítésre kerül.
- $Value(C_1) \geq Cost(G) \geq Value(C_2)$: Ekkor csak C_1 fogyasztó használhatja a generátort, hiszen C_2 értéke kisebb, mint a generátor használati "költsége". Ebben az esetben felmerülhet, hogy C_2 fogyasztó nem minden periódusban kapja meg a szükséges áramellátást.
- $Cost(G) \geq Value(C_1) \geq Value(C_2)$: Itt sem C_1, sem pedig C_2 nem használhatja a generátort.

Fontos, hogy a kisebb prioritású fogyasztók összértéke ne haladhasson meg egy magasabb prioritású fogyasztói értéket! (A prioritás száma minél kisebb, annál fontosabb, vagyis az 1-es a legfontosabb prioritású.) Az érték megadásához ezért az alábbi képlet ajánlott:

$$Value = 1000 * \frac{1}{2^{x-1}}, \text{ahol } x = priority \tag{4.44}$$

A P-gráf modellben az áramszükséglet a maximális, és nem a szükséges, vagyis minimális paraméterénél jelenik meg a fogyasztóknál. Az érték miatt mindenképpen a lehető legtöbb áramot próbálja szolgáltatni a modell, de nem kerül megoldhatatlan állapotba, amennyiben nem tudja ezt teljesíteni. További kérdés lehet az akkumulátorok használata fogyasztói prioritás függvényében. Amennyiben a modell nem engedi meg egy fogyasztónak, hogy az akkumulátorokat is használhassa, abban az esetben a fogyasztót, mint terméket és az áramot, mint köztes terméket összekapcsoló műveleti egység kapacitásának felső korlátja meg kell, hogy egyezzen az aktuális periódus megújuló forrást használó termelői által előállított áram mennyiséggel. A modell célja az, hogy a lehető legtöbb igény kerüljön kielégítésre. A célfüggvény tehát:

$$max\Big(\sum_{i=1}^{N}\Big(\sum_{j=1}^{|C|}Value(C_i^j)-\sum_{j=1}^{|G|}Cost(G_i^j)\Big)\Big). \tag{4.45}$$

A modell teljes körű generálásához az előbbiekből a következő input adatokra van szükség: $Cost(G)$, R_i^j, C_i^j, $Priority(C^j)$, C^j fogyasztó energia ellátásának engedélyei (például használhat-e generátort vagy akkumulátort), B_{init}^j, $Capac\_B^j$, $Storage\text{-}Amort(B^j)$, $LoadAmort(B^j)$, $UsageAmort(B^j)$, ahol i jelöli a periódust, j pedig az egységek beazonosítására szolgál. Ezen adatok ismeretében a fenti egyenletek alapján bármennyi periódusra megadható a pontos modell. Minden periódus közepén az energia (E_i), mint köztes anyag áll. Ehhez kerül bekötésre egy műveleti egységen keresztül az R_i^j, valamint a B_i^j. A B_i^j és E_i közötti műveleti egységből kiinduló él súlya $1-UsageAmort(B^j)$ lesz. A gráfhoz kell adni a töltést és tárolást szimbolizáló műveleti egységeket. B_i^j-ből a tárolóba, E_i-ből pedig a töltésbe fog él vezetni. Ezekhez minden periódusban tartozik egy kapacitás is, mint nyersanyag, amiből egy-egy él vezet ezen műveleti egységekbe. Amennyiben az első periódus kerül modellezésre, úgy az akkumulátorok kezdeti töltöttségét (B_{init}^j) egy nyersanyag fogja reprezentálni, ami egy műveleti egységen keresztül bekötésre kerül B_i^j-hez. Amennyiben nem az első periódus, úgy az előző periódus tárolójából és töltéséből (amik műveleti egységek) a megfelelő amortizációval csökkentett éleket kell vezetni B_i^j-be. A periódusnál nyersanyagként fog megjelenni a drágább, nem megújuló energiát használó berendezés által szolgáltatott energia. Ezt követően minden fogyasztó, mint termék fog megjelenni, a termék ára pedig a fent kiszámított $Value$ lesz. E_i-ből egy-egy él fog vezetni a termékekhez tartozó műveleti egységekbe, ahonnan szintén él vezet a termékekhez. Ha a fogyasztó csak a megújulót használhatja, úgy ennek a műveleti egységnek a felső korlátja meg fog egyezni a megújuló által termelt energia (R_i^j) mennyiségével. Amennyiben egy fogyasztó használhatja a generátort, úgy szükséges egy-egy műveleti egységet hozzáadni a gráfhoz, majd G_i-t ezzel, illetve ezt C_i^j-vel összekötni. Minden fogyasztónak külön-külön szükséges egy ilyen műveleti egység a generátorhasználathoz. Az utolsó ciklusban a tárolóból és az akkumulátor töltésből egy-egy él vezet a termékként reprezentált maradó töltöttségi igénybe, aminek alsó korlátja az igényelt mennyiség.

A bemeneti adatok alapján könnyedén generálható egy ilyen, multiperiódusos modell. Ebben nyújt segítséget a 18. algoritmus. Az egyik bemenet a *Megujulok*, ami több megújuló forrást használó termelőt is magában foglalhat. Egy-egy ilyen termelőnél minden egyes periódusra meg kell adni, hogy mennyi áramot tud előállítani. Ez akár az időjárás-előrejelzés alapján, számítással is történhet. A másik szükséges információ a *Fogyasztok* ismerete. Ez szintén egy halmaz, ami a különböző fogyasztókat foglalja magában. Egy-egy fogyasztó rendelkezik prioritással és minden periódusra megvan, hogy mennyi az áram-igénye. A 18. algoritmusnál észrevehető, hogy prioritással nem, viszont fogyasztói értékkel számol. Ez az érték általában a prioritástól függ, és egy javasolt módszer a kiszámítására a 4.44. egyenletben található. A *Generator* szintén tartalmaz egy értéket. Ez nem az üzemanyagár, hanem a használatának a "költsége", vagy más néven "büntetése". Ha értéke magasabb, mint a legmagasabb prioritású fogyasztó prioritásból számolt értéke, úgy egyik fogyasztó sem használhatja. A generátort csak azon fogyasztók használhatják, akinél a kiszolgálásának a számított értéke magasabb, mint a generátor értéke. Mivel ez a különböző eseteknél eltérhet, így célszerű, ha az algoritmus külön bekéri. Szükséges még ismerni a rendszerhez tartozó akkumulátorokat is, amik az *Akkuk* halmazban vannak. Egy-egy akkumulátor rendelkezik kapacitással, információról arra vonatkozóan, hogy mennyi a kezdeti töltöttsége, illetve mennyi töltöttségnek kell maradnia az utolsó periódus után, valamint különböző amortizációs értékekkel, százalékban kifejezve: töltési-, használati-, és tárolási amortizáció.

Motivációs példa:

A fenti leírás segítségével bármilyen hosszú, és bármekkora felbontású (periódus hosszára értve) esetet lehet modellezni. Bár lehet, hogy elsőre egyszerűnek tűnik az áramelosztás kiszámítása, hosszabb időszak esetén az korántsem triviális. A megoldó hatékonyságának szemléltetésére egy esettanulmány került kidolgozásra a 4.4. táblázatban található paramétereknek megfelelően. A példa két fogyasztót tartalmaz különböző prioritásokkal, akiknek az igényét egy naperőmű és egy nem túl jó állapotú akkumulátor biztosítja. A 4.26. ábrán látható az igények előrejelzése és a naperőmű várható termelése fél órás bontásban. Az KKV, vagyis a kis és közepes vállalkozás energiaigényének előrejelzése a [61] alapján, a háztartások igényének előrejelzése pedig a [62] hivatkozás alapján történt. A naperőmű adatai saját mérések alapján, felskálázással kerültek meghatározásra. Mindhárom adat a valóságnak megfelel és összehasonlítható, még akkor is, ha nem ugyanazon a helyen és időben került megállapításra. A modellben elsőként a KKV igényének kielégítése a cél, hiszen magasabb prioritású, a háztartások csak ezután juthatnak áramhoz. Generátor használata erősen elkerülendő (a struktúra épp ezért nem is tartalmazza), az elektromosságot csak és kizárólag a naperőmű és az akkumulátor biztosítja, ameddig tudja.

A naperőmű mérete úgy lett skálázva, hogy a KKV igényét magas rendelkezésre állás mellett ki tudja elégíteni. Ez azt vonja magával, hogy többször is lesz extra termelés. Az extra termelés elosztásának stratégiái és a modell jósága a másodlagos prioritású fogyasztónak juttatott áram alapján mérhető és összehasonlítható, hiszen az elsődleges fogyasztó igényei mindig kielégítésre kerülnek. Egy általános, heurisztikus áramelosztási stratégia alkalmazásakor az akkumulátorok töltöttségének szintjétől teszik függővé a másodlagos fogyasztónak juttatott áramot. Extrém esetben a háztartás csak akkor kaphat energiát, ha a magasabb prioritású fogyasztók teljesen ki vannak elégítve és az akkumulátorok teljesen fel vannak töltve. Az esettanulmányban a másodlagos fogyasztónak a szükséglete 114,393 kWh. Ha ez a fogyasztó csak akkor kaphat áramot, ha az akkumulátorok teljesen töltöttek, akkor összesen 9,163 kWh energiát fog csak kapni, ami

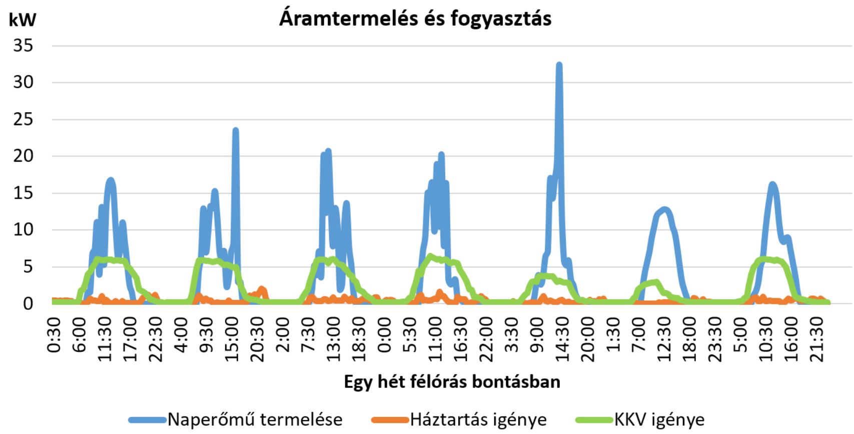
Algoritmus 18: Periodikus termeles elorejelzes es fogyasztoi igenyek $\Rightarrow$ multiperiodusos PNS

input : Megujulok, Fogyasztok, Akkuk, Generator
output: ($\mathcal{P}^m$,$\mathcal{R}^m$,$\mathcal{O}^m$) multiperiodusos modell

for $i = 1; i \leq |Fogyasztok_1 \rightarrow igeny|; i++$ **do**
    var szumMegujulobol = 0;
    for $r \in Megujulok$ **do**
        $\mathcal{R}^m := \mathcal{R}^m \cup megujulo_i^r(U = r \rightarrow termeles_i)$;
        $\mathcal{O}^m := \mathcal{O}^m \cup megujulobol_i^r(\{megujulo_i^r\}\{e_i\})$;
        $szumMegujulobol+ = r \rightarrow termeles_i$;
    end
    $\mathcal{R}^m := \mathcal{R}^m \cup generator_i(cm = Generator \rightarrow ertek)$;
    for $c \in Fogyasztok$ **do**
        $\mathcal{P}^m := \mathcal{P}^m \cup fogyaszto_i^c(U = c \rightarrow igeny_i, cm = c \rightarrow ertek)$;
        $\mathcal{O}^m := \mathcal{O}^m \cup fogyasztonak_i^c(\{e_i\}\{fogyaszto_i^c\})$;
        if $c \rightarrow csakMegujulobol$ **then**
            $fogyasztonak_i^c(U = szumMegujulobol)$
        end
        $\mathcal{O}^m := \mathcal{O}^m \cup generatorbol_i^c(\{generator_i\}\{fogyaszto_i^c\})$;
    end
    for $b \in Akkuk$ **do**
        $\mathcal{R}^m := \mathcal{R}^m \cup akkuKapac_i^b(U = b \rightarrow kapacitas)$;
        if i=*1* **then**
            $\mathcal{R}^m := \mathcal{R}^m \cup akku_1^b(U = b \rightarrow kezdetiToltottseg)$;
        end
        $\mathcal{O}^m := \mathcal{O}^m \cup akkubol_i^b(\{akku_i^b\}\{(100 - (b \rightarrow hasznalatiVeszteseg))/100 * e_i\})$;
        if $i = |Fogyasztok_1 \rightarrow igeny|$ **then**
            $\mathcal{P}^m := \mathcal{P}^m \cup marad_i^b(L = b \rightarrow maradjon)$;
            $\mathcal{O}^m := \mathcal{O}^m \cup toltes_i^b(\{e_i, akkuKapac_i^b\}\{(100 - (b \rightarrow toltesiVeszteseg))/100 * marad_i^b\})$;
            $\mathcal{O}^m := \mathcal{O}^m \cup tarolas_i^b(\{akku_i^b, akkuKapac_i^b\}\{(100 - (b \rightarrow tarolasiVeszteseg))/100 *$
              $* marad_i^b\})$;
        else
            $\mathcal{O}^m := \mathcal{O}^m \cup toltes_i^b(\{e_i, akkuKapac_i^b\}\{(100 - (b \rightarrow toltesiVeszteseg))/100 * akku_{i+1}^b\})$;
            $\mathcal{O}^m := \mathcal{O}^m \cup tarolas_i^b(\{akku_i^b, akkuKapac_i^b\}\{(100 - (b \rightarrow tarolasiVeszteseg))/100 *$
              $* akku_{i+1}^b\})$;
        end
    end
end

Periódusok:	Egy hét félórás bontásban	
Fogyasztó # 1:	KKV	1-es prioritású
Fogyasztó # 2:	Háztartás	2-es prioritású
Megújuló:	Naperőmű	
Akkumulátor:	Kapacitás: 75/2 kWh	Kezdő töltés: 75/2 kWh
	Tárolási veszteség: 2% /fél óra	
Diesel generátor:	jelenleg elérhetetlen	

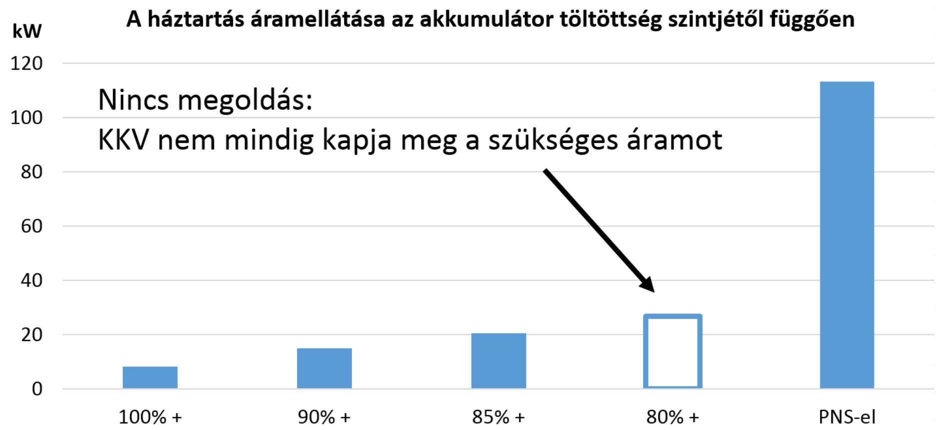
4.4. táblázat. A motivációs példa paraméterei



4.26. ábra. 168 órára becsült áram igények és termelés

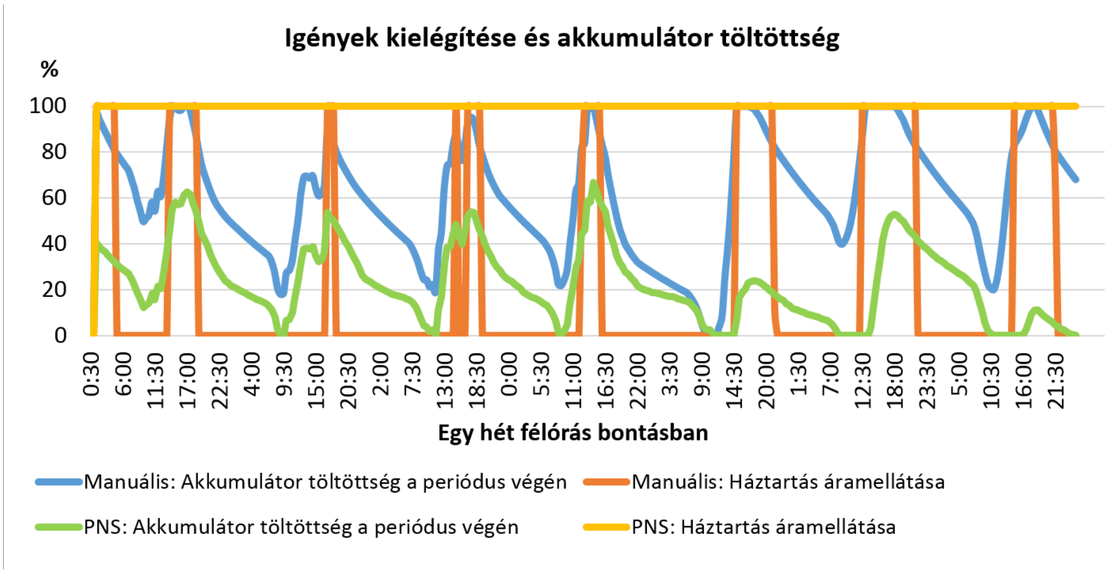
elhanyagolható mennyiség.

Ez a mennyiség növelhető, ha a háztartás már az előtt kaphat áramot, hogy az akkumulátorok töltöttségi szintje elérné a 100%-ot. Például, ha a másodlagos fogyasztók már 90%-tól kezdhetnék használni az akkumulátort, akkor a 90% feletti részt használhatnák fel. Kiszámítva ekkor 18,561 kWh jut nekik. Ezt a határt csökkentve növelhető a háztartásoknak juttatott energia mennyisége, de megvan annak a kockázata, hogy végül emiatt nem lehet majd kielégíteni teljes mértékben a magasabb prioritású fogyasztót. Figyelve erre megállapítható, hogy ez a határ 83%-nál van az esettanulmányban. Ha a háztartás már 82%-tól használhatná az akkumulátort, akkor egy későbbi időpontban a tárolt energia már nem lesz elég az KKV-nek. Ez látható a 4.27. ábrán is.



4.27. ábra. Háztartás áramellátása

A fent bemutatott P-gráf alapú megközelítésnek köszönhetően a több periódusos optimalizálás előre vetíti az energia rendelkezésre állását a későbbi periódusok során, amit kihasználva növelhető az áramellátás. Ennek eredményeképpen kiszámíthatóvá válik, mikor kaphat a háztartás áramot anélkül, hogy a tervezési horizont bármelyik időszakában hatással lenne a KKV ellátására. A 336 periódusosból álló P-gráf modell generálása és megoldása után az eredmények alapján a háztartás összesen 114,393 kWh-ot kap, ami az igényeinek 100%-át fedezi. Ez szintén látható a 4.27. ábrán.



4.28. ábra. A fogyasztói igények kielégítése és az akkumulátor töltöttsége

A 4.28. ábra talán még jobban illusztrálja az eredményeket. Látható, hogy bár heurisztikus megoldásnál is van, hogy a háztartás az igényének megfelelően 100%-osan megkapja az áramot, de ez csak ritkán fordul elő, míg a folyamathálózat-szintézissel mindig elég energia jut neki a teljes időszak alatt. Az ábra továbbá azt is jól megmutatja, hogy ez az akkumulátor kiegyensúlyozott használatának köszönhető. Heurisztikus megoldásnál az akkumulátor töltöttsége az idővel felhalmozódik a KKV ellátásától függetlenül. Ezzel szemben a PNS biztosítja, hogy az akkumulátorban elég töltöttség legyen a KKV számára, az extra töltöttséget pedig kiszolgáltatja a háztartásoknak.

Az optimalizálási módszer segítségével további kérdések kerülhetnek megválaszolásra, például, hogy szükséges vagy elégséges-e az adott kapacitású akkumulátor, vagy, hogy hogyan befolyásolja a kezdeti akkumulátor töltöttség a későbbi elosztást. A kérdésekre a válaszokat egyfajta érzékenység vizsgálat adja meg. A 4.5. táblázatból kiderül, hogy ha 75/2 kWh kapacitású akkumulátor szerepel a modellben, és az kezdetben teljesen fel van töltve, úgy a fogyasztói igények 100%-ban kielégíthetők. Ha viszont kezdetben nincs feltöltve, sem a KKV, sem pedig a háztartás igényeit nem lehet teljes mértékben kielégíteni. Mivel a KKV igényeit mindenképpen ki kell elégíteni, ezért szükséges egy generátor használata. Látható, hogy a generátorból 14,508 kWh energiát kell kapni, hogy megfelelően biztosítva legyen az ellátás. Ezt az első periódusokban kapja a KKV, egészen addig, amíg a naperőmű már nem biztosít elegendő áramot.

Ha az akkumulátor kapacitása 60/2 kWh-ra csökken, de teljes töltéssel indul, a háztartás igényei csak részben kerülnek kielégítésre, vagyis egy ennél kicsivel nagyobb akkumulátorra lesz szükség. 65/2 kWh már megfelelő lenne az igények teljes kielégítéséhez. Ahhoz, hogy a teljes ellátás garantálva legyen, 20,176/2 kWh kezdeti töltöttségnek kell lenni. További két eset lett még megvizsgálva: kisebb akkumulátor kapacitás egy generátor bevezetése mellett, ahol az egyik esetben csak a KKV, a másik esetben pedig mindkét fogyasztó használhatja a generátort. Azt eldönteni, hogy megéri-e kisebb akkumulátort beszerezni, és néha generátorral kiegészíteni a termelést, már a döntéshozók feladata.

Kezdeti töltés (kWh)	Kapacitás (kWh)	Generátor (kW)	KKV kap (kWh)	Háztartás kap (kWh)
75/2	75/2	unavailable	742.710	114.393
0/2	75/2	unavailable	728.201	109.129
0/2	75/2	14.508, only SME	742.710	109.129
60/2	60/2	unavailable	742.710	109.572
65/2	65/2	unavailable	742.710	114.393
20.176/2	65/2	unavailable	742.710	114.392
50/2	50/2	3.046, only SME	742.710	102.618
50/2	50/2	14.821, both	742.710	114.393

4.5. táblázat. Optimális áramellátás különböző konfigurációk mellett

A fenti példa és a modell egy, az egyetemmel konzorciumban lévő vállalat igénye kapcsán született. Elmondható, hogy a való életben több problémánál is felmerül a multiperiódusos jelleg, ahol akár több száz periódusra nézve kell elvégezni az ütemezést. A fogyasztók megkülönböztetése prioritás szempontjából is sok helyen felmerülhet. Ezeket a P-gráf modelleket bár grafikusan már nem lehet megjeleníteni, de ez nem is cél, hiszen a döntéshozók számára sokszor ismeretlen a P-gráf felépítése. Az eredményeket ezért érthető formában kell visszaadni. A bemutatott modell alkalmas arra, hogy kezelje a fogyasztói prioritásbeli különbségeket, képes több száz periódussal is számolni, a fogyasztási-, és időjárási előrejelzések alapján egy optimális ütemezést visszaadni, valamint a microgrid rendszerekbe is beágyazható. A korábbi, heurisztikus megoldáshoz képest a modellel jelentősen növelhető a háztartásoknak juttatott energia, illetve előnye, hogy minimálisra csökkenti a túltermelést amellett, hogy a fogyasztói elégedettséget növeli. Az erőforrások egyenletes kihasználása miatt a környezetre gyakorolt káros hatások is csökkenthetők. Egy ilyen megoldó alapjául szolgálhat az üzemanyag beszerzés ütemezésének, hiszen látszik, hogy mikor és mennyi üzemanyagra lesz szükség. A későbbiekben a modell kibővíthető az összevont periódusos igényekkel (például 3 periódus alatt kell egy bizonyos mennyiséget szolgáltatni), vagy több forgatókönyv segítségével akár kockázatbecslés is végezhető [63].

4.6. A fejezet rövid összefoglalása

A fejezetben a folyamathálózat-szintézisen belül multiperiódusos modellekkel foglalkoztam. Megmutattam, hogy a periódusok közötti tárolás hogyan képes a terhelést elosztani a periódusok között, ezáltal erőforrásokat és költséget spórolva. Mivel korábban csak esettanulmányokban jelent meg a multiperiódusos modell, szükségét éreztem egy általános modell elkészítésének. Az alapmodellt változó és nem-változó részekre bontva a többperiódusos modell algoritmikusan generálható. Általános modellt adtam a multiperiódusos modellek létrehozására, valamint a periódusok közötti tárolás megvalósítására. Ezeket az általános modelleket nem csak algoritmikusan dolgoztam ki, de szoftveresen is megvalósítottam és integráltam ezt a modellgenerálási lehetőséget a P-graph Studioba. A fejezet második felében különböző alkalmazhatósági területeken szemléltettem a multiperiódusos modellek relevanciáját, úgy, mint a gyártástervezés, a hulladékgazdálkodás vagy a karbantartási idők. Kiemelten vannak jelen a dolgozatban a különböző energiatárolást megvalósító multiperiódusos modellek. Az egyik ilyen, egy ipari együttműködés keretében felmerült megújuló forrásból származó energiaelosztási probléma, ahol különböző prioritású szereplők igényeit kell kielégíteni az időjárás-előrejelzések és akkumulátor töltöttségek figyelembe vételével. A bemutatott modell alkalmazásával jelentősen növelhető a fogyasztók igényeinek kielégítése.

4.6.1. A fejezethez tartozó tézis

Kidolgoztam egy általános modellt a folyamathálózat-szintézis multi-periodikus leírásához és a periódusok közötti tárolók megvalósításához.

(a) Megállapítottam, hogy a multiperiódusos modellek változó és nem-változó részekre bonthatók, melyek modell szinten is megkülönböztethetőek. Egy multiperiódusos modell ezek alapján algoritmikusan generálható a változó részek paramétereinek megadásából.

(b) Mivel a legtöbb gyártási folyamat rendelkezik valamilyen tárolási lehetőséggel, ezért a multiperiódusos modell leírást kiterjesztettem a periódusok közötti tárolók bevezetésével. A tárolók megvalósítására szintén egy általános modellt dolgoztam ki.

(c) Algoritmust adtam arra, hogyan kell generálni az egyperiódusú modellből több periódusból álló és tárolókat tartalmazó modellt.

(d) Speciális feladattípusokon keresztül mutattam be az általános multiperiódusos modell paraméterezésének lehetőségeit, úgy mint a gyártásütemezés vagy az energiaelosztás, ahol ipari együttműködés keretében, valós esettanulmányokban is alkalmazásra került a módszer.

4.6.2. A fejezet témaköréhez kapcsolódó publikációk

Nemzetközi folyóiratcikk

- Bertók Botond és Bartos Anikó: Renewable energy storage and distribution scheduling for microgrids by exploiting recent developments in process network synthesis, Journal of Cleaner Production, 2019. (IF = 6,395) [64]
- Bertók Botond és Bartos Anikó: Algorithmic process synthesis and optimisation for multiple time periods including waste treatment: Latest developments in P-graph Studio software, folyóirat: Chemical Engineering Transactions, 70. szám, 97-102. oldal, 2018. [33]

Nemzetközi konferencia előadások

- Bartos Anikó, Bertók Botond és Szlama Adrián: Optimal design of multi-period process networks including storages for renewable resources, konferencia: International Congress on Sustainability Science Engineering, Balatonfüred, Magyaroroszág, 2015. [65]
- Bartos Anikó és Bertók Botond: P-graph Framework: Computer Aided Model Generation and Solution for Supply Network Optimization Problems, konferencia: European Working Group on Location Analysis Meeting 2015., Budapest, Magyarország, 2015. [66]
- König Éva, Bartos Anikó és Bertók Botond: Free Software for the Education of Supply Chain Optimization, konferencia: VOCAL Optimization Conference: Advanced Algorithms 2016., Esztergom, Magyarország, 2016. [67]
- Bartos Anikó és Bertók Botond: Estimation of the return of investment in new technologies regarding periodically changing demands, availability of resources, and storages, konferencia: Chemical Engineering Days 2017, Veszprém, Magyarország, 2017. [68]
- Bartos Anikó és Bertók Botond: Software for Economical Evaluation of Utilizing Periodically Available Renewable Resources, konferencia: SPIL 2017 (Energy, water, emission, & waste in industry an cities), Brno, Csehország, 2017. [69]

5. fejezet

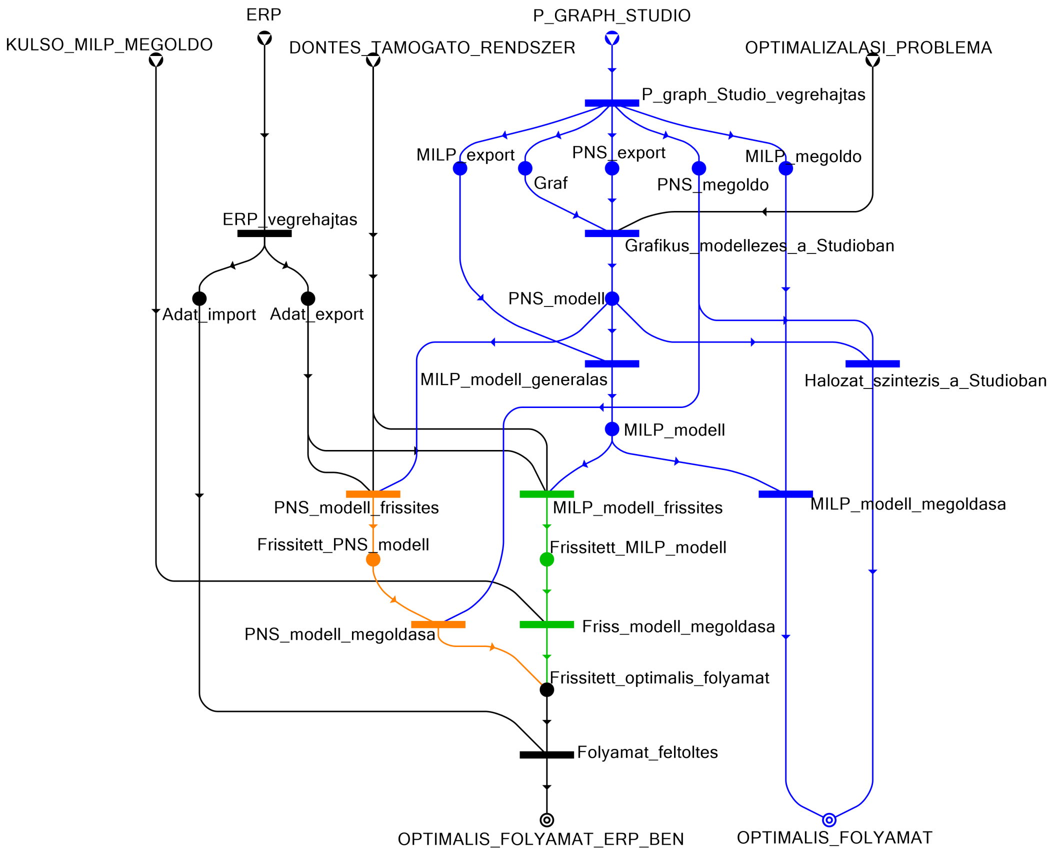
Line Balancing

Közép- és Kelet-Európában az ipar egy jelentős részét teszik ki a különböző összeszerelő üzemek. Az autóipari és elektronikai gyárakban a termékeket kész alkatrészekből, gyártósor vagy szalag mellett szerelik össze főként emberi erőforrást használva. Tipikusan, különböző összeszerelési lépéseket kell egymás után végrehajtania a betanított munkaerőnek, és bár ez a fajta munkavégzés nem igényel speciális tudást vagy szakképzést, és rövid időn belül bárki képes beletanulni, mégis a gyárak jelentős munkaerőhiánnyal küzdenek. Az üzemekbe érkező megrendelések teljesítése szempontjából fontos, hogy a vállalat a lehető legtöbbet hozza ki az erőforrásaiból, ezért lényeges a munkaerő-feladat optimális, azaz egyenletes elosztása. A szakirodalom ezt a típusú optimalizálást gyártósor kiegyensúlyozásnak, vagy *line balancing*-nak hívja, és ott érdemes igazán alkalmazni, ahol tömegtermelés folyik, és egy adott típusú terméket hosszabb ideig gyártanak ugyanazon a soron [70]. (Például egy napig ugyanazt a típusú számítógépet állítják elő.)

Bár gyártószalagot elsőként Chicagói húsüzemek alkalmaztak, mégis Ford tette ismertté 1913-ban, és már ezekben a korai években is megpróbálták ütemezni az általuk való termelést [71]. 1955-ben Salveson azonosította az egyensúlyi késleltetést, vagy elvesztegetett időt, mint minimalizálandó célt és már lineáris programozási feladatokként közelítette meg a problémát [72]. Az optimalizálás mind lokális, mind pedig globális szinten cél volt, és kiterjedt nem csak az összeszerelés megvalósítására, de a gépek és a munkaerő fizikai elhelyezésére is. Az évekkel és a gyárak fejlődésével egyre több helyen vezettek be optimalizálási megoldásokat, amik a gyártószalagok elhelyezésétől és gyártott termékek variációjától is függtek [73].

Ezek közül a legelterjedtebbek az úgynevezett egysoros modellek, ahol egy terméket gyártanak, a vegyes összeszerelő sorok, ahol több termék is megjelenik a gyártósoron, a multi-modellek, ahol átállás is várható, és a kétoldali gyártósorok, ahol a szalag mindkét oldalán találhatók szerelőállomások [74, 75]. Az eladásokat tekintve olyan megoldások is születtek, ahol a túltermelés büntetésként plusz költséget vont maga után, valamint léteznek sztochasztikus és heurisztikus megoldások is [76]. Ebben a fejezetben egy egyszerű egysoros modell kerül bemutatásra, ahol az elvégzendő feladatok sorrendje kötött, továbbá ennek megvalósítása P-gráffal, ahol nem csak az optimális, de az N-legjobb eredmény is megkapható. A fejezet végén bemutatásra kerül egy elkészült szoftver, illetve annak felhasználó-oldali visszajelzése.

5.1. Probléma meghatározás

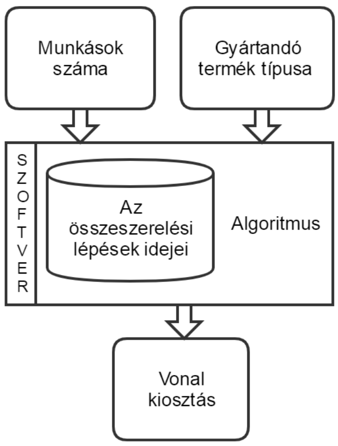


5.1. ábra. Döntéstámogatás PNS-sel

A bemutatásra kerülő vonal-kiegyensúlyozási problémánál különböző típusú számítógépeket szerelnek össze egyszerű, egysoros szalag mellett a már rendelkezésre álló alkatrészekből egy magyarországi összeszerelő üzemnél. Bár több, különböző típusú termék összeszerelése folyik a gyárnál, mégis vannak olyan szerelési lépések, amik megegyeznek az egyes termékeknél. A gyártástervezés szakaszában azt határozzák meg, hogy melyik terméket mikor szereljék össze a gyártósoron, aminek alapjául a megrendelési idők, határidők és prioritások szolgálnak, valamint az, hogy az összeszereléshez szükséges összes alkatrész rendelkezésre áll-e. A termelésvezető feladata meghatározni azt, hogy a beérkezett munkaerőből ki milyen feladatot végezzen a nap folyamán a már kiválaszott, legyártandó terméken, és ezzel együtt törekednie is kell arra, hogy ezek a feladatok lehetőleg egyenletesen legyenek szétosztva közöttük. Ez az elosztás a szalag melletti összeszerelést tekintve optimális, ha minden munkás nagyjából ugyanannyi ideig foglalkozik egy termék összeszerelési lépéseivel ez által minimalizálva a ciklusidőt. A gyakorlatban ez a kiosztás manuálisan, intuíciókon alapulva történt. Az előre tervezés is nehézkes, ugyanis az, hogy pontosan hány munkást lehet beállítani dolgozni, csak az adott műszak elején derül ki, ezért az ütemezést és beosztást minden műszak elején újra meg kell csinálni a maximális hatékonyság elérése érdekében. A folyamatos és lehetőleg minél gyorsabb tervezés miatt a gyárnak szüksége volt egy optimalizáló szoftverre, ami segíti ezt az elosztást.

A vállalati döntéseket nagy mértékben képes támogatni a folyamathálózat-szintézis, és az 5.1. ábrán látható folyamatábrából jól kiolvasható, hogy hogyan képes beépülni a vállalatirányítási rendszerekbe. Erre kétféle megközelítést lehet alkalmazni: vagy egy általános MILP modell kerül felírásra, és frissítésre, és azt kell megoldani akár egy általános MILP megoldóval, vagy a PNS modell kerül frissítésre egy külön modullal és végül azt kell megoldani a P-gráf megoldóval. Utóbbi esetben, vagyis amikor csak a modell kerül felírásra, el lehet tekinteni a grafikus reprezentációtól. Ekkor a modell megoldására elegendő csak a P-graph Studio alatt is megtalálható P-gráf megoldó szoftver.

A cél egy olyan modell és ezzel együtt szoftver elkészítése volt, ahol a legyártandó termék típusának és a munkába állítható alkalmazottak számának tudatában a kimenet egy sorkiosztás, ami mellett a ciklusidő a lehető legkisebb. Fontos megjegyezni, hogy itt az összeszerelési lépések sorrendje kötött. A munkások munkaállomásokhoz vannak rendelve, és egy műszakon belül nem változtatják a munkaállomásukat, vagyis egy munkás csak egy adott, egymás utáni lépéssorozatot képes végrehajtani egy terméken. Ez a fent látható kétféle megközelítésben is bemutatásra kerül, vagyis mind P-gráf modellel, mind pedig egyszerű MILP felírással. A szoftver felépítése az 5.2. ábrán látható.



5.2. ábra. A szoftver sematikus felépítése

Formálisan ez, vagyis a gyártósor kiegyensúlyozási probléma a következőképp adható meg:

$$objective function = min\{cycle\} \tag{5.1}$$

$$cycle = max\{w_1 t, ..., w_i t, ..., w_n t\} \tag{5.2}$$

ahol $w_i t$ a az összes, szereléssel töltött ideje az i-edik munkásnak és n pedig az aznapi munkások száma, akikre kioszthatóak a feladatok. Egy munkás összes, szereléssel töltött idejét az alábbi képlet adja meg:

$$w_i t = \sum_{j=k}^{l} s_j \tag{5.3}$$

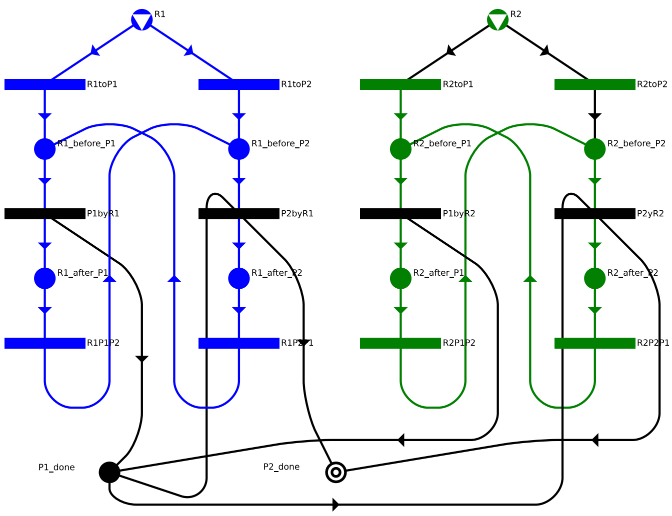
$$m = |S| \tag{5.4}$$

$$\nexists j : 1 \leqslant k \leqslant j \leqslant l \leqslant m, s_k \in w_i S, s_l \in w_i S, s_j \notin w_i S \tag{5.5}$$

$$\forall s_j \in \cup w_i S : w_i S \subseteq S \tag{5.6}$$

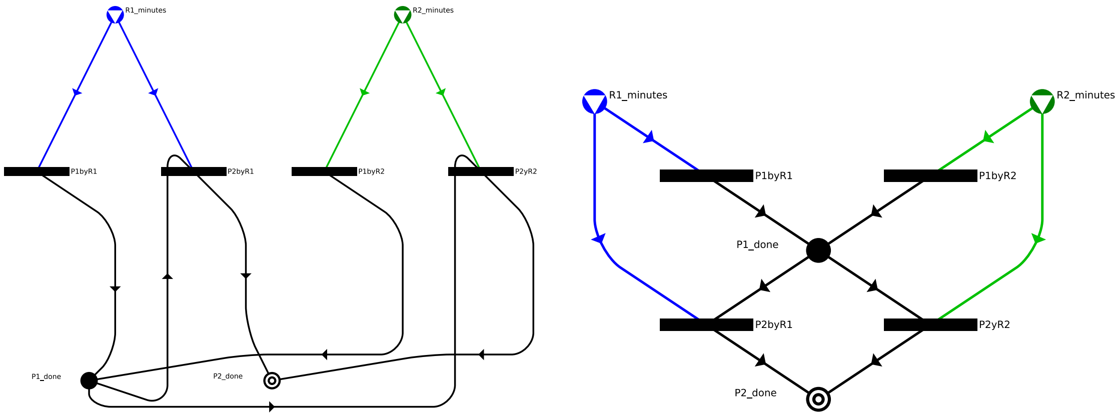
ahol k az első és l az utolsó részfeladat, amit a w_i munkás végez, és s_j a j-edik részfeladat ideje. S jelöli a részfeladatok halmazát. Az optimalizáláshoz szükséges ismerni az összes összeszereléshez szükséges részlépést és azok idejét. Ezeket az összeszerelési időket előre le kell mérni minden egyes termék esetén, és mátrixos formában eltárolni. Ez a későbbiekben bővíthető, amennyiben új termék gyártásába kezdene a vállalat, de az optimalizálás során rejtve marad a felhasználó elől.

5.2. Modellezés és megoldás a TCPNS egy egyszerűsített változatával



5.3. ábra. Két taszkos két berendezéses ütemezési feladat

Ahogy arról a 3. fejezetben is szó volt, az ütemezési feladatok jól megoldhatók időkorlátos folyamathálózat-szintézis feladatként. A gyártósor kiegyensúlyozás is egy ilyen, ütemezési probléma, ami a kötött sorrendje és a lineáris jellege miatt a standard modell egy egyszerűsített változataként is felírható. A transzformáció végigvezetéséhez legyen a kiindulóstruktúra az 5.3. ábrán látható, két taszkból és két berendezésből álló ütemezési feladat. Mindként berendezés képes mindkét taszk végrehajtására, és az első taszk a második előfeltétele. A második taszk elkészültével a termék is létrejön.



5.4. ábra. Az elhagyható részek eliminálása és átrendezés

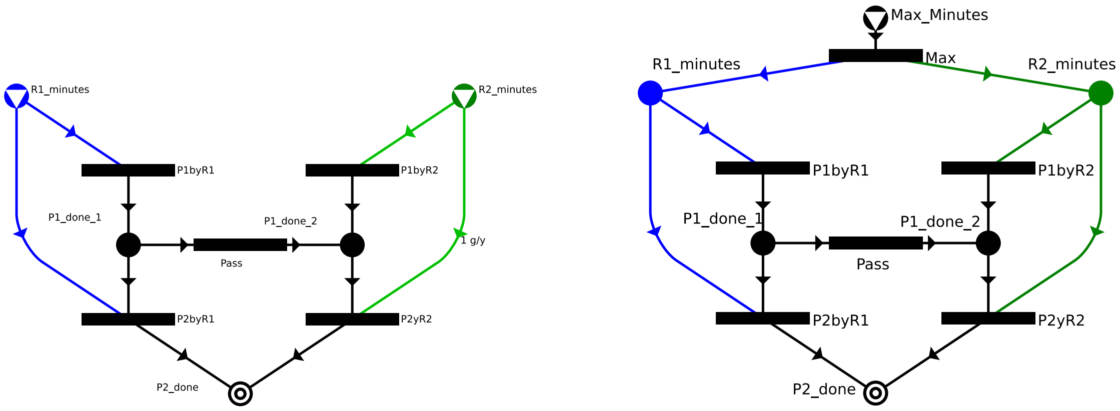
Line balancing feladatoknál a berendezések az egyes emberek lesznek, akik képesek az összeszerelésre, a taszkok pedig az összeszerelés egy-egy lépései. Mivel az összeszerelés lépéseinek sorrendje kötött és a dolgozóknak egymás utáni lépéseket kell végrehajtaniuk, így nincs szükség a visszafele váltásokra. Az általános struktúrából ezek az élek és műveletek elhagyhatók lesznek.

Arra sincs szükség, hogy az idő minden műveletnél mérve legyen, hiszen egy ilyen, lineáris folyamatnál az összidő az eltelt idővel lesz egyenlő. A gyártósor kiegyensúlyozás szempontjából felesleges elemek eltávolításával létrejött struktúra az 5.4. ábrán, bal oldalon látható. Az ábra jobb oldala pedig ugyanaz, csak egy átrendezés után. Mivel meg kell tudni különböztetni, hogy melyik munkás melyik összeszerelési lépést végezte el, vagyis az ábrán látható köztes anyagot melyik műveleti egység állította elő, ezért szükséges azt szétbontani több részre.

Egy munkás egy félkész termékkel két dolgot tehet: vagy elvégzi a következő összeszerelési lépést, vagy átadja a soron következő dolgozónak azt. Az átadás megvalósítására be kell vezetni egy műveleti egységet. Ez minden köztes anyagnál, és minden egymást követő munkásnál szerepelni fog. A kibővített modell az 5.5. ábrán, bal oldalon látható.

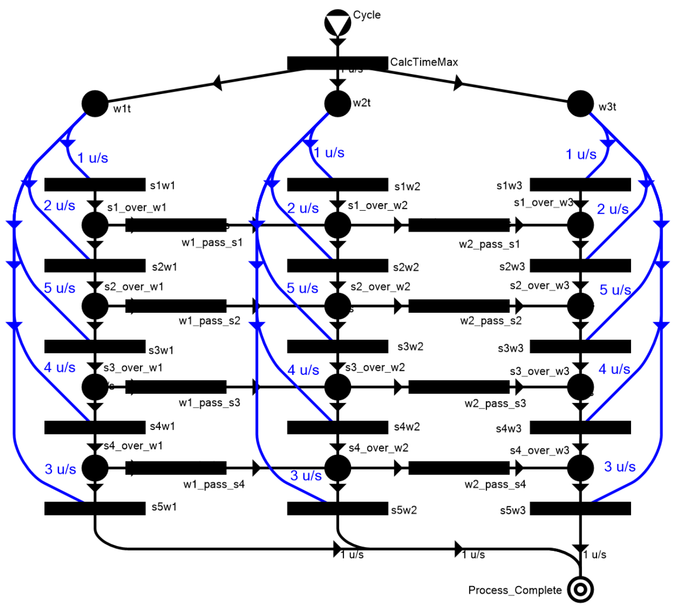
Ahhoz, hogy a ciklusidő mérhető legyen, szükséges bevezetni egy ezt reprezentáló nyersanyagot, valamint köztes anyagokként az egyes munkások idejét. Ez utóbbiból kiinduló súlyozott élek segítségével adható meg, hogy melyik munkafolyamat mennyi időt vesz igénybe. Ahhoz, hogy végül kiderüljön, mennyi a ciklusidő, vagyis az a leghosszabb idő, amit egy munkás összeszereléssel tölt, be kell vezetni egy-egy műveleti egységet minden munkás ideje fölé, és ebbe bevezetni a nyersanyagként megjelenő ciklusidőt.

A módosított gráf az 5.5. ábrán, jobb oldalon látható. A ciklusidőt az újonnan bevezetésre



5.5. ábra. A struktúra kiegészítése

került nyersanyagra állított költség fogja megadni megoldáskor. Ahhoz, hogy modellszinten a gyártás végbemenjen a költség ellenére is, a termék minimum és maximum mennyiségét 1-re kell állítani. Így lehet megkapni, hogy mennyi lesz egy termék ciklusideje. El kell kerülni, hogy a modell részmunkával számoljon, olyan értelemben, hogy egy összeszerelési lépést ne bontson kisebb részekre. Ehhez a műveleti egységek alsó korlátját szintén 1-ben kell meghatározni, vagyis vagy egy teljes részmunkát végez az egység, vagy semmit. A modellt megoldva a költség megmondja, hogy mennyi lesz egy termék ciklusideje, azaz a leghosszabb idő, amit egy munkás szereléssel tölt.



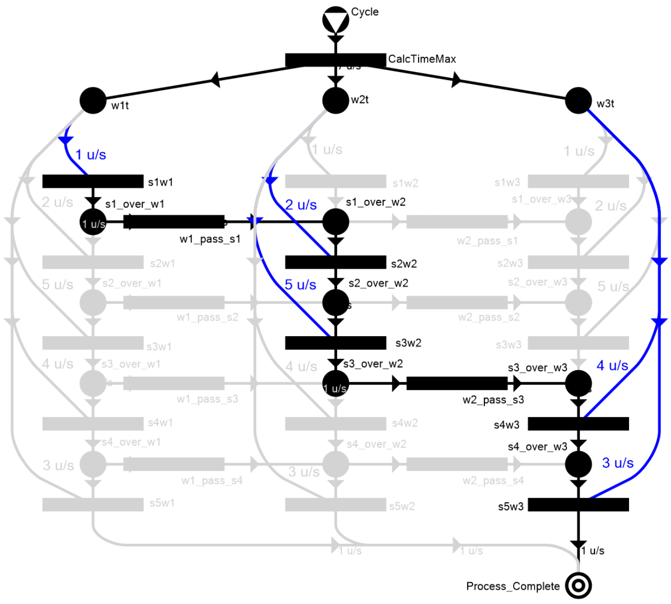
5.6. ábra. Szemléltető példa három alkalmazottal és öt részfeladattal

Valamennyivel bonyolultabb, de még mindig egy szemléletes, áttekinthető példa látható az 5.6. ábrán, ahol három dolgozó végez egy öt lépésből álló szerelési folyamatot. Az egyes lépések idejei az alábbiak:

Felpakolás	Kicsomagolás	Összecsavarozás	Lezárás	Lepakolás
1 s	2 s	5 s	4 s	3 s

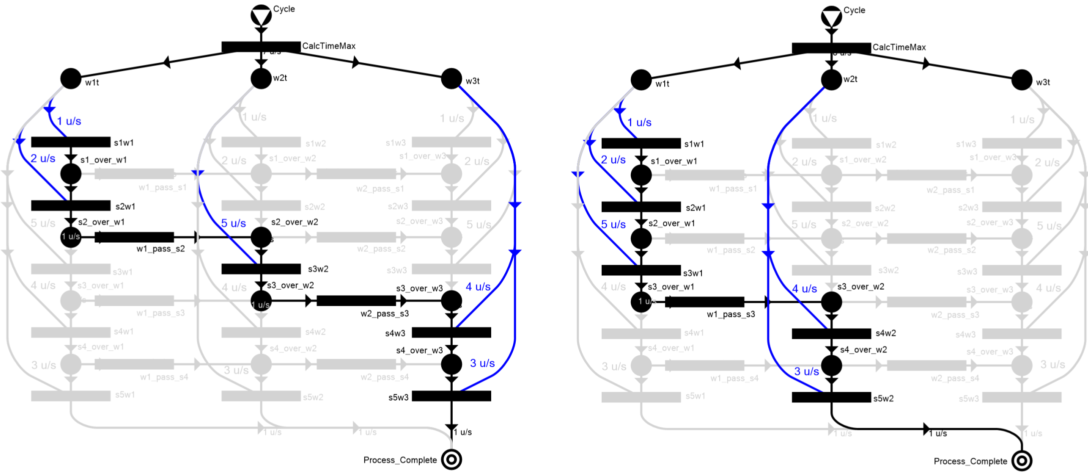
5.1. táblázat. Szerelési lépések idejei - példa

Az ábrán egy-egy sor megfeleltethető egy-egy szerelési lépésnek, míg az oszlopok a humánerőforrást reprezentálják. Jól látható, hogy minden lépésnél lehetséges a félkész termék átadása a következő munkás számára, illetve, hogy a részmunkák idejei súlyozott élekként jelennek meg (kék színnel jelölve). A feladaton az RCABB algoritmust futtatva az 5.7. ábrán látható struktúra jön ki eredményül, mint legjobb megoldás, ahol a ciklusidő 7 másodperc, és a második és harmadik dolgozó is 7-7 másodperc alatt végez a részfeladataival. A legjobb esetben az első dolgozó csak az első részfeladatot végzi, míg a második és harmadik munkás két-két, egymás utáni műveletet hajt végre. Ideális esetben minden munkás $(\sum_{j=1}^{m} s_j)/n$ időt töltene szereléssel, de általában ilyen leosztás nem valósítható meg. Az elérhető ciklusidő a munkások létszámával fordítottan arányos.



5.7. ábra. Szemléltető példa legjobb megoldása

A modell második legjobb megoldásánál szintén 7 másodperc a ciklusidő, viszont ezt más leosztás eredményezi. A harmadik legjobb megoldásnál a ciklusidő 8-ra emelkedik, de ekkor az összeszerelés lépései három helyett két munkásra terhelődnek. Ebből következik az is, hogy két



5.8. ábra. Szemléltető példa második és harmadik legjobb megoldása

munkás esetén a ciklusidő 8 másodperc. A második és harmadik legjobb megoldás az 5.8. ábrán látható. A munkások számának növelésével kiderül, hogy 4 munkással már 5 másodperc alatt is teljesíthető az összeszerelés, de ez már nem csökkenthető tovább, hiszen a leghosszabb részfeladat ideje 5 másodperc. Ezek az időigényesebb részlépések nagyobb ciklusidőt okozhatnak.

A P-gráf erősségeinek kiaknázásához ebben az esetben nem szükséges a grafikus reprezentáció, elég csak a matematikai modell megvalósítása. Mivel egy jól leírható problémáról van szó, a modellgenerálás automatikus lehet, amennyiben ismert a munkások száma és a részfeladatok idejei. Ehhez segítséget nyújt a 19. algoritmus, ami a bemeneti gyártósor-kiegyensúlyozási adatok alapján előállít egy P-gráf modellt.

5.3. A probléma MILP modellje

A gyártósor kiegyensúlyozás megfogalmazható lineáris programozási feladatként, ahol az optimum értéket a lineáris függvények szélén kell keresni és ahol a feltételek lineáris egyenlőtlenségekként jelennek meg. Számos ingyenesen elérhető MILP megoldó létezik már a piacon, így célszerű lehet ilyen irányból is megadni a line balancing problémát. A MILP modellben változóként fog megjelenni a ciklusidő (C^T), a munkások idejei (w_t), valamint két mátrix: melyik munkás melyik tevékenységet végzi ($\mathbf{W^P_{i,j}}$), és ki kinek adja át a feladatot, ha végzett ($\mathbf{W^A_{i,j}}$). $\mathbf{W^P_{i,j}}$ egy bináris mátrix, ahol '1'-es értékként jelenik meg, hogy melyik dolgozó melyik részfeladatot fogja végezni. $\mathbf{W^A_{i,j}}$ szintén bináris mátrix, amely megadja, hogy a munkás kinek adja át a félkész terméket, amikor az összes részfeladatával végzett. n jelöli a munkások számát, m pedig az elvégzendő feladatok számát.

$$\begin{array}{rrcl} & C^t & \geq & 0 \\ \forall i \in \{1..n\}, \forall j \in \{1..m\}: & W^P_{i,j}, W^A_{i,j} & \in & \{0,1\} \end{array} \tag{5.7}$$

Algoritmus 19: Line balancing $\Rightarrow$ PNS

input : W, S
output: $\mathcal{P}, \mathcal{R}, \mathcal{O}$ parameteres szintezis modell

$\mathcal{P}$:=ProcessComplete(L=U=1);
$\mathcal{R}$:=Cycle(cm = 1\$);
forall $i \in W$ **do**
    CalcTimeMax:=({Cycle}{w_it});
end
$\mathcal{O}$:=$\mathcal{O}\cup$ CalcTimeMax;
forall $i \in W$ **do**
    forall $j \in S$ **do**
        if $j = 1$ **then**
            S_jW_i:=({$s_j * w_it$ }{ s_joverw_i}, L=1);
        else
            if $j = s_m$ **then**
                S_jW_i:=({$s_j * w_it$, $s_{j-1}overw_i$ }{$ProcessComplete$},L=1);
            else
                S_jW_i:=({$s_j * w_it$, $s_{j-1}overw_i$ }{ s_joverw_i },L=1);
            end
        end
        $\mathcal{O}$:=$\mathcal{O} \cup S_jW_i$;
        if $i \neq w_n$ **then**
            w_ipasss_j:=({s_joverw_i }{ s_joverw_{i+1}});
            $\mathcal{O}$:=$\mathcal{O} \cup w_ipasss_j$;
        end
    end
end

A cél a ciklusidő minimalizálása:

$$\min \quad C^t \tag{5.8}$$

Korlátként az alábbiak jelennek meg: Minden részfeladatot szét kell osztani a munkások között.

$$\forall j \in \{1..m\}: \quad \sum_{i=1}^{n} W_{i,j}^{P} \quad = \quad 1 \tag{5.9}$$

Minden munkásra igaz, hogy a ciklusidő $\geq$ mint a munkás összmunkaideje, amit az általa elvégzett részmunkák idejének összege ad. Ez kezdetben nullával kerül inicializálásra, majd ehhez adódik hozzá az elvégzett feladatok ideje.

$$\begin{aligned} \forall i \in \{1..n\}: \quad w_i^t &\leq C^t \\ \forall i \in \{1..n\}: \quad w_i^t &= \sum_{j=1}^{m} s_j * W_{i,j}^P \\ W_{1,1}^P &= 1 \end{aligned} \tag{5.10}$$

$\mathbf{W^P}$ mátrixot, ami azt jelöli, hogy melyik munkás melyik feladatot fogja végezni, és amiből látható, hogy melyik lesz az utolsó feladata, szintén korlátozni kell annak érdekében, hogy a munkások egymásnak adják majd a félkész terméket anélkül, hogy egy lépést még kisebb részlépésekre bontanának, illetve garantálni kell, hogy csak egymást követő lépéseket tudjanak végrehajtani.

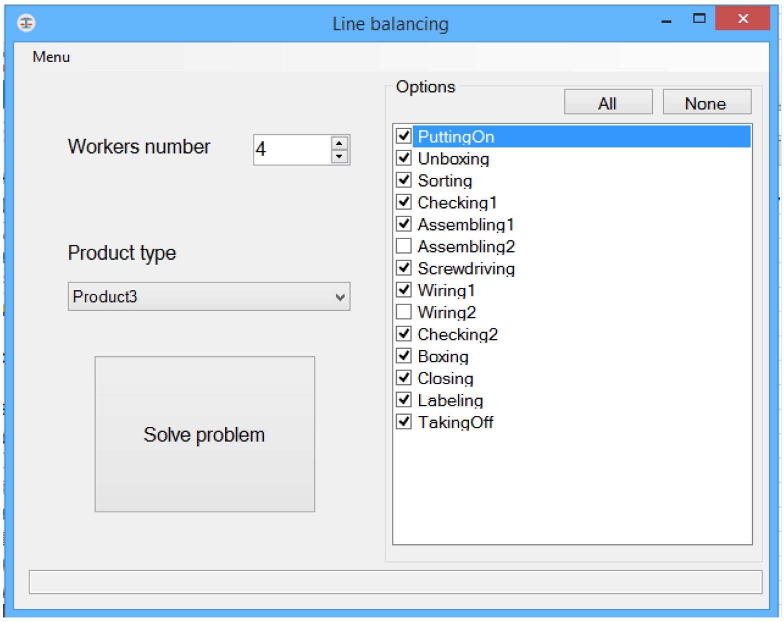
$$\begin{aligned} \forall i \in \{2..n\}, \forall j \in \{2..m\}: \quad & W_{i,j-1}^P + W_{i-1,j-1}^A \\ = \quad & W_{i,j}^P + W_{i,j-1}^A \end{aligned} \tag{5.11}$$

$$\forall j \in \{2..m\}: \quad W_{1,j-1}^P = W_{1,j}^P + W_{1,j-1}^A \tag{5.12}$$

Egy mindenki számára elérhető ingyenes megoldó vegyes egész értékű feladatokhoz a COIN-OR CBC [77, 78]. Amennyiben a fenti leírás alapján létrehozott .lp fájl a COIN-OR CBC-vel megoldásra kerül, az eredmény, vagyis a ciklusidő megtalálható lesz egy szöveges, bárki számára könnyen olvasható fájlban. Az eredmények a P-gráf megoldóval összhangban vannak, eltekintve attól, hogy ezek a megoldók csak a legjobb, míg a P-gráf algoritmusai az N-legjobb megoldást adják vissza. Mivel mind MILP, mind pedig P-gráf megközelítésben egy jól meghatározható modell írja le a feladatot, ezért azt könnyű automatizálni, valamint egy olyan szoftvert létrehozni, ahol grafikus felületen megadhatók a főbb adatok, és ami az eredményeket akár a laikusok számára is érhető formában tudja visszaadni.

5.4. Szoftveres megvalósítás és visszajelzések

Mivel a termelésvezetők -vagyis akik eldöntik, hogy az aznapi munkára fogható emberek közül ki milyen feladatot végezzen- nem akarnak vagy nem tudnak matematikai modelleket felírni minden nap az optimalizálás érdekében, ezért szükséges egy olyan szoftver létrehozása, ami a lehető legegyszerűbb módon képes generálni egy optimális sorkiosztási tervet. Egy ilyen létrehozott szoftver képe az 5.9. ábrán látható. A felhasználó egy legördülő menüből egyszerűen ki tudja választani a gyártandó termék típusát, és egy számok megadására szolgáló mezőben megadhatja, hogy hány munkásra kívánja szétosztani a feladatokat. A részidőket tartalmazó táblázat - mivel ennek a módosítása nem szükséges minden egyes optimalizálásnál-, a menü alatt érhető el. Ez a táblázat szabadon szerkeszthető, vagyis, ha új terméket készül bevezetni a vállalat, annak lemért idejeivel egyszerűen kiegészíthető, vagy a már nem gyártott termékek adatai törölhetőek. A szoftver indításakor a táblázat beolvasásra kerül, és a termékek nevei ezek alapján az adatok alapján kerülnek a lehulló menübe. A "Solve problem" gombra kattintva legenerálódik és megoldásra kerül a modell, aminek eredménye a döntéshozók számára is könnyen olvasható táblázatba íródik ki.

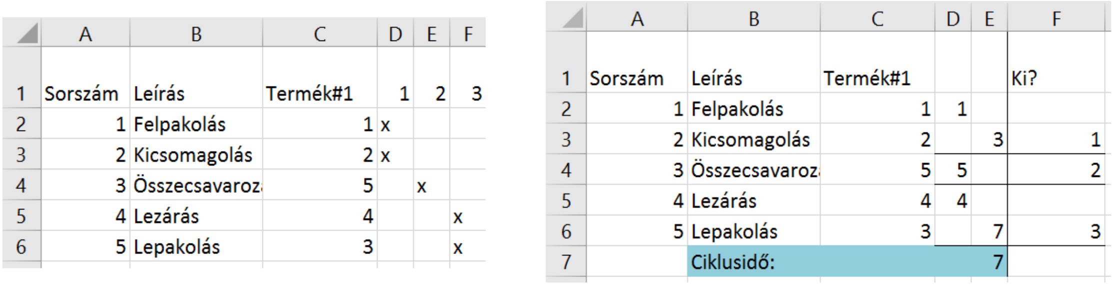


5.9. ábra. A szoftver felépítése

A szoftver egy magyarországi összeszerelő üzemmel konzultálva készült és valós életbeli, éles adatokkal lett tesztelve. A vállalat az együttműködés során jelezte, hogy bár vannak olyan termékei, melyeknek nevei és azonosítói megegyeznek, mégis, eltérő lépésekből állhat az összeszerelésük. Ez a megrendelések egyéni igényei miatt van, így megoldást kellett találni arra, hogy hogyan lehet úgy megadni egy terméket, hogy közben az egyes lépések opcionálisak maradjanak. Mivel az *alternatíváknál* is kötött a sorrend, vagyis, az összes lehetséges, egyéni szerelés unióját véve a szerelésnek megadható egy sorrendiség, ezért csak azt kell megadni, hogy az összes lépésből melyik szükséges ténylegesen az aktuális szereléshez. Az időket tartalmazó mátrixban ezért egy egyéni jelölést bevezetve megadható, hogy mely lépések fixek az egyes típusoknál és melyek lehetnek opcionálisak. A szoftver fő ablakában a típus kiválasztása után az opcionális lépések felsorolásra kerülnek, így a gyártásvezető könnyedén ki tudja választani az éppen relevánsakat, és azok alapján végezheti el az optimalizálást. A vállalattól kapott input adat 21 különböző termék lépéseit tartalmazta, ahol minimum 48, legfeljebb pedig 128 lépésből állt egy-egy szerelés.

Az 5.10. ábrán a szoftver kimeneti felépítése látható Excel formátumban, két munkalapon. Ez az 5.6. ábra megoldása is egyben. Jól látható, hogy melyik munkás melyik feladatot kell, hogy végezze, illetve a szerelési lépésekkel hogyan nő egy-egy munkás ideje. A gyártásvezető ezek alapján már ki tudja osztani az aznapi feladatokat a beosztottainak. A vállalattól pozitív visszajelzés érkezett a szoftverre, ahol az eredmények magukért beszélnek. A korábbi, manuális sorfelállítással szemben szoftveres segítséggel akár 20-25%-nyi hatékonyságnövekedést is el tudtak érni. Egy ilyen összehasonlítás látható az 5.11. ábrán.

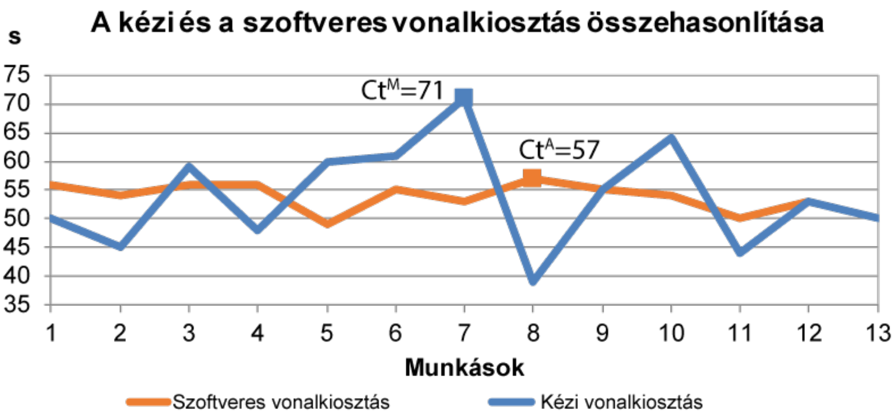
A szoftver lehetőségei tovább növelhetők a párhuzamosított munkavégzés bevezetésével, vagy-



	A	B	C	D	E	F
1	Sorszám	Leírás	Termék#1	1	2	3
2	1	Felpakolás	1	x		
3	2	Kicsomagolás	2	x		
4	3	Összecsavaroz	5		x	
5	4	Lezárás	4			x
6	5	Lepakolás	3			x

	A	B	C	D	E	F
1	Sorszám	Leírás	Termék#1			Ki?
2	1	Felpakolás	1	1		
3	2	Kicsomagolás	2		3	1
4	3	Összecsavaroz	5	5		2
5	4	Lezárás	4	4		
6	5	Lepakolás	3		7	3
7		Ciklusidő:			7	

5.10. ábra. Példa a kimenetre



5.11. ábra. A manuális és szoftverrel segített sorkiosztás összehasonlítása

is, ha egy nagyobb részfeladatot akár két emberre is ki lehet osztani. A probléma megoldása az egyszerűtől a kevésbé triviálisakig is terjedhet, hiszen felmerül a kérdés, hogy hány embert lehet párhuzamosítani, vagy hány részre lehet egy nagyobb feladatot bontani. A nagy részfeladatok miatt bekövetkező ciklusidő növekedésre ez egy megoldást kínálhat.

Összegzésként elmondható, hogy egy olyan modellt, majd erre építve szoftvert sikerült létrehozni, amit a magyarországi összeszerelő üzemek is felhasználhatnak hatékonyságnövelés érdekében. Mivel ezekben a gyárakban egyre gyakoribb a munkaerőhiány, így fontos, hogy folyamataik optimalizálva legyenek, és a rendelkezésre álló erőforrásokat a lehető leghatékonyabban használják ki. Az elkészült szellemi termékeket és szoftvert egy összeszerelő üzemmel való közös kooperációnak köszönhetően sikerült valós adatokon is tesztelni. Az átadást követően a visszajelzések alapján azt a mindennapi tervezés során, a gyakorlatban is használják, jelentős hatékonyságnövekedést elérve vele.

5.5. A fejezet rövid összefoglalása

A fejezetben számos magyarországi nagyvállalatot is érintő témakörrel, a gyártósor kiegyensúlyozással foglalkoztam. A feladat arra keresi a választ, hogy a gyártósor mellett dolgozók közül ki melyik részfeladatot lássa el ahhoz, hogy a leggyorsabban, vagyis a legrövidebb ciklusidővel létrejöjjön egy termék. A problémát általános, ütemezési feladatként közelítettem meg, majd a modellt a feladat speciális jellegéből adódóan redukáltam és egészítettem ki, melyből egy általános, mennyiségkorlátos hálózatszintézis feladat modellt alkottam meg. A fejezetben ismertettem a probléma általános MILP modelljét, végül bemutattam a modell szoftveres implementációját, melyet valós piaci adatokon teszteltem. A modell eredményességét alátámasztja a vállalatnál elért $\sim$ 20%-os ciklusidő-csökkenés.

5.5.1. A fejezethez tartozó tézis

Létrehoztam egy modellt a gyártósor kiegyensúlyozás problémához a korábban PNS-el megvalósított általános ütemezés alapján.

(a) Módszert adtam a "line balancing", vagyis a gyártósor kiegyensúlyozás ütemezési feladatként való felírására.

(b) Levezettem, hogy a gyártósor kiegyensúlyozási probléma ütemezési feladatát időkorlátos hálózatszintézis feladatként, majd annak egyszerűsítése után mennyiségkorlátos hálózatszintézis feladatként hogyan lehet megfogalmazni.

(c) A modell és a hozzá készült döntéstámogatási rendszerbe építhető szoftver hatékonyságát és eredményességét valós piaci adatokon teszteltem.

5.5.2. A fejezet témaköréhez kapcsolódó publikáció

Nemzetközi folyóiratcikk

– Bartos Anikó és Bertók Botond: Production line balancing by P-graphs, folyóirat: Optimization and Engineering, 1-18. oldal, 2019. (IF = 1,824) [11]

6. fejezet

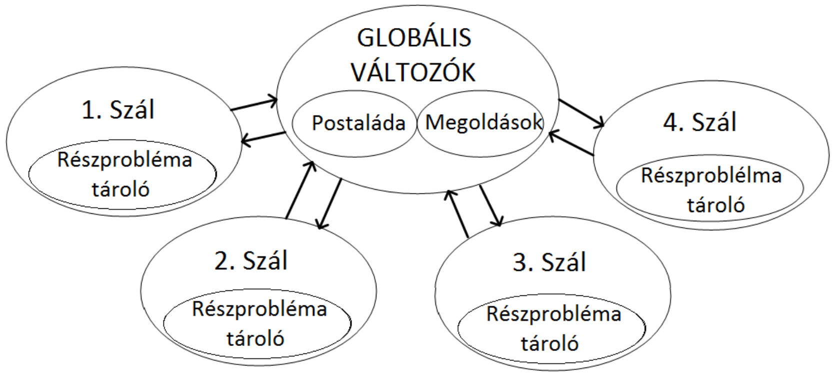
A hálózatszintézishez kapcsolódó algoritmus fejlesztések

Ahogy arról már a 3. fejezetben is szó volt, nem csak modellezési szempontok mentén lehet optimalizálni egy algoritmust, de programozástechnikailag is jelentős hatékonyságnövekedés érhető el, például párhuzamosítással. A Branch & Bound algoritmusok párhuzamosítására már számos megoldást kínál a szakirodalom. A P-gráf egyik fontos algoritmusának, az RCABB-nek egy korábbi változatához már elkészült egy párhuzamosított megoldás, de az nem a CPU magjain, hanem külön processzorokon valósította meg a párhuzamos eljárást. A modern számítógépek mindegyike már több maggal rendelkezik, ezért pazarlás lenne ezt nem kihasználni, főleg nagyobb számításidejű problémáknál. Ebben a fejezetben bemutatásra kerül egy több magon megvalósított Branch & Bound algoritmus, valamint annak paraméteroptimalizálása.

6.1. Az RCABB algoritmus párhuzamosításának megvalósítása CPU-n

A párhuzamosítás igényel némi módosítást az RCABB algoritmuson. Ebben az esetben nem csak egy központi tároló van, ahol a megoldások és részproblémák megosztásra kerülnek, hanem minden szál rendelkezik saját tárolóval is a részproblémái számára. Ez a struktúra a 6.1. ábrán látható. Amennyiben egy szál saját részprobléma tárolója kiürül, vagyis minden részproblémát már kifejtett, úgy egy kérést küldhet a többi szál felé. Egy központi helyen -nevezzük postaládának- vannak a globális változók, amik minden szál számára elérhetőek. Ez a postaláda szolgál többek között a részproblémák megosztására is.

Az algoritmus indulásakor az egyik szál a kezdeti problémát a saját tárolójába menti, ahol elkezdi kifejteni, megoldani azt. A többi szál ekkor még csak vár, illetve elküldött már egy-egy kérést a postaládába részprobléma igénylésre. Ha a kezdeti szál saját részprobléma tárolójában a részproblémák száma elér egy előre meghatározott szintet, illetve van igény a többi szál felől



6.1. ábra. A szálak közötti adatok megosztása

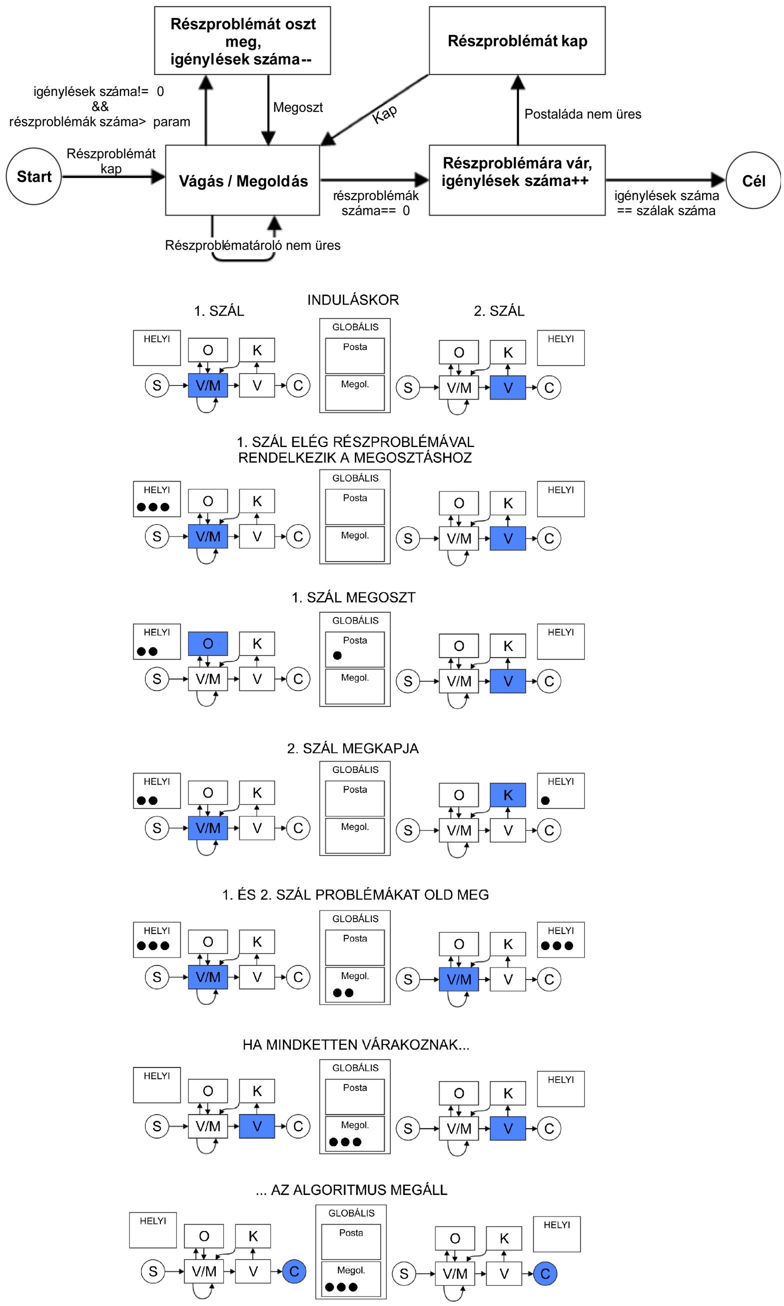
részproblémára, úgy a szál a saját tárolójából áttesz egyet a postaládába. Áttételkor a részprobléma a saját tárolóból törlődik. A várakozó szálak közül a leggyorsabb észreveszi, hogy a postaládában megosztásra került egy részprobléma, ezért átteszi a saját tárolójába, majd elkezdi kifejteni azt. Ilyenkor a postaládából a részprobléma törlődik. Fontos, hogy egyszerre csak egy szál módosíthassa a postaláda tartalmát. A részproblémák megosztása és elvitele a szálak között nem prioritás alapú, hanem mindig az a szál fog cselekedni, amelyik leghamarabb észlelte az igényt vagy megosztást.

A globális változók között, a megosztott részproblémákon kívül megtalálhatóak a megoldások és ezzel együtt az aktuális korlát értéke is. A szálak mindig csak akkor oszthatnak meg részproblémát, ha a saját tárolójukban a részproblémák száma meghalad egy előre meghatározott szintet, valamint van igény a többi szál részéről részproblémára. A szálak nem csak indulásakor kerülhetnek várakozó állapotba, hanem akkor is, ha már minden részprobléma kifejtésre került a saját tárolójukból. A várakozó szálak folyamatosan kérdezik le a postaládát, hogy van-e benne megosztott tartalom, amit megkaphatnak. A keresés akkor ér véget, ha a részprobléma-igények száma egyenlő a szálak számával, vagyis minden lokális tároló üres, és a postaládában sincsen megosztott részprobléma. Minden szál viselkedése azonos, nincs megkülönböztetett szerepű közöttük. A 6.2. ábra felső részén látható folyamatábra illusztrálja az egyes szálak viselkedését a végrehajtás közben. Az ábra alsó részén egy példaműködés látható két szálon történő futtatáskor.

A 6.3. ábrán látható egy, a már párhuzamosított megoldó futtatását szemléltető keresőfa, ahol az egyes színek azt reprezentálják, hogy melyik szál dolgozta fel az adott csomópontot. Jól látható, hogy a szálak terheltsége közel azonos. A párhuzamosítás után elvégzett tesztelések eredményei azt mutatták, hogy a párhuzamosítással jelentős mértékben megnövekedett a hatékonyság.

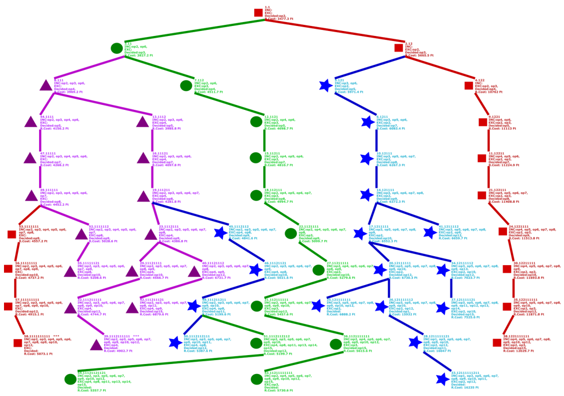
6.2. Teszthalmaz készítése

Ahhoz, hogy az algoritmusok hatékonyságát megfelelően lehessen mérni, elengedhetetlen, hogy legyen egy teszthalmaz, amely nagy számú, különböző méretű és bonyolultságú problémát tartalmaz. 1998-ban Bertók és társai kifejlesztettek a P-gráfhoz egy probléma generátort, ami számos



6.2. ábra. Állapotdiagram és kommunikáció két szál között

konfigurációs lehetőséggel rendelkezik, és képes létrehozni a P-gráf formátumának megfelelő problémát [79]. Egy konfigurációs fájlon keresztül többek között beállítható a műveleti egységek, a nyers- és köztes anyagok, a termékek és melléktermékek száma, amit egy diszkrét eloszlást meg-



6.3. ábra. Részfák feldolgozása szálak szerint

Elem	Minimum szám	Maximum szám
Műveleti egység	9	90
Nyersanyag	1	45
Köztes anyag	7	116
Termék	1	5
Él	35	479

6.1. táblázat. Teszthalmaz tulajdonságai

valósító függvény biztosít. Beállítható, hogy a műveleti egységeknek átlagosan mennyi input és mennyi output anyaga lehet, valamit egyéb topológiát befolyásoló tényezők is, úgy mint a hurkok száma.

A P-gráf probléma generáló szoftver segítségével elkészült egy 150 tesztfájlt tartalmazó problémahalmaz, a 6.1. táblázatban látható tulajdonságokkal. (A részletes tulajdonságok a mellékletben találhatóak.) Ezek a fájlok online is publikálásra kerültek, hogy mások számára is segítséget nyújthassanak hasonló tesztelésekhez [80]. A párhuzamosított algoritmus és annak paraméteroptimalizálása is ezeken a fájlokon került tesztelésre.

6.3. Paraméteroptimalizálás

Az RCABB algoritmus párhuzamosításának megvalósítása során felmerült néhány kérdés: Milyen gyakran ellenőrizzék a szálak a tárolót, hogy ne vegyen el túl sok időt, de ne történjen felesleges számítás információhiány miatt? A szálak az általuk kifejtésre kerülő keresőfa melyik részéről osszanak meg a többiekkel részproblémát? Minimum hány megoldás kell, hogy maradjon a saját tárolóban megosztás után? Ezek, és maga a szálak száma nyitott beállításként kerültek implementálásra. A párhuzamosított algoritmus a következő paraméterbeállításokkal bír:

– Szálak száma: Az első paraméter a szálak száma, ami, amennyiben értéke ’1’, úgy a többi paramétert feleslegessé is tudja tenni, hiszen akkor a párhuzamosítás jelentőségét veszti.

Ez a szám azt mondja meg, hogy hány szál dolgozzon a részproblémákon egyszerre. A paraméter alapérzelmezett értéke: *MAX*, vagyis az összes lehetséges szálon dolgozzon.

– Megosztási stratégia: Ezzel azt lehet megadni, hogy a szálak igény esetén a saját tárolójuk elejéről vagy végéről adjanak-e részproblémát a postaládába. Vizuálisan elképzelve ezek a részproblémák a keresőfa felsőbb illetve alsóbb részein helyezkednek el. *"GlobalNext"* stratégiánál a keresőfa felső részéről történik a megosztás, hiszen ez a teljes keresőfát nézve globálisan a következő kifejtendő részfeladat. *"LocalNext"* stratégiakor a megosztás egy alsóbb szintről történik. A paraméter alapérzelmezett értéke: 75% LocalNext, 25% GlobalNext.

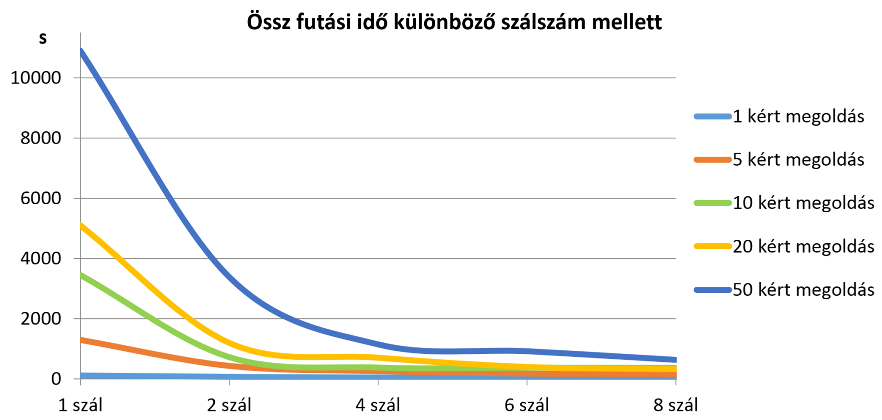
– Ellenőrzési gyakoriság: Ezzel azt lehet beállítani, hogy milyen sűrűn (hány ciklusonként) történjen meg a postaláda ellenőrzése. Ez részprobléma megosztásnál és igénylésnél is szerepet játszik. A paraméter alapérzelmezett értéke: '1', vagyis minden ciklusban ellenőriz.

– A saját tárolóban maradó részproblémák minimum száma: Azt mondja meg, hogy minimum hány részprobléma kell, hogy maradjon - vagy más megközelítésben legyen a megosztás után- a tárolóban ahhoz, hogy az megoszthasson közülük egyet igény esetén. A paraméter alapérzelmezett értéke: '1', vagyis 1 megoldás maradjon legalább a saját tárolóban.

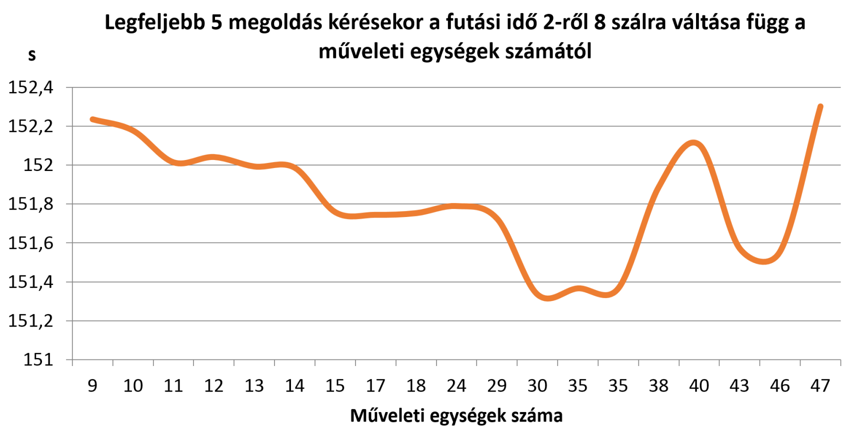
Ezek a paraméterek képesek befolyásolni a párhuzamos megoldó viselkedését abban az esetben, ha a szálak száma több, mint egy, ezáltal növelve a hatékonyságot. Vagyis kijelenthető, hogy az összes paraméter függ a szálak számától. Mivel a többi paraméter sem feltétlenül független egymástól, ezért az optimalizálás során először az alap beállításokhoz képest egyesével történt a paraméterhangolás, majd a keresési tér leszűkítése után együtt.

A következőkben bemutatott konfigurációk mindegyike az előző fejezetben említett 150 mintapéldán lett tesztelve 1, 5, 10, 20 és 50 legjobb megoldást kérve. Minden teszt három alkalommal lett futtatva a pontosabb eredmények érdekében, vagyis több, mint 25000 alkalommal futott a megoldó a paraméteroptimalizálás során. A legjobb paraméterbeállítás meghatározásán túl felmerült a kérdés, hogy vajon függ-e a probléma strukturális tulajdonságától vagy sem az, hogy mi lesz a legjobb beállítás, illetve milyen tulajdonságokat is kéne ehhez figyelembe venni.

Szálak száma: Az első paraméter a szálak száma, ami 1, 2, 4, 6 és 8 szálon került tesztelésre. Az eredmények a várakozásoknak megfelelően kimutatták, hogy a szálak számával a hatékonyság is növekszik. A 6.4. ábrán a számítási összidők láthatók különböző szálszám és különböző kért megoldásszámok mellett, azonban ha a kért megoldásszám kevesebb, mint 5 és a feladat maximum 30 műveleti egységet tartalmaz, úgy két szálon futtatni a leghatékonyabb. Ez az eredmény abból következik, hogy kis, gyorsan megoldható feladatoknál feleslegesen időt veszítünk a szálak közötti kommunikációval. Két szál viszont még mindig hatékonyabban tudja megoldani a feladatot ebben az esetben is, mint egy szál. A 6.5. ábra megmutatja, hogy 5 megoldás kérésekor nagyjából 30 műveleti egységnél érdemes váltani két szálról a maximálisan elérhetőre.



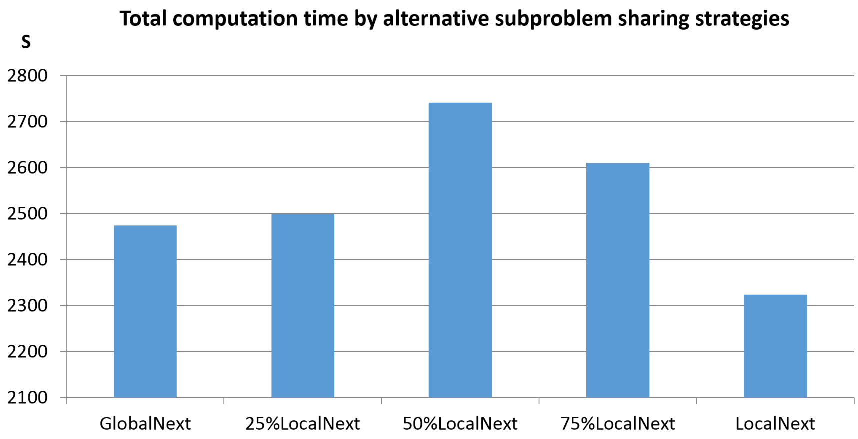
6.4. ábra. A számítási összidők összehasonlítása különböző számú szálak mellett



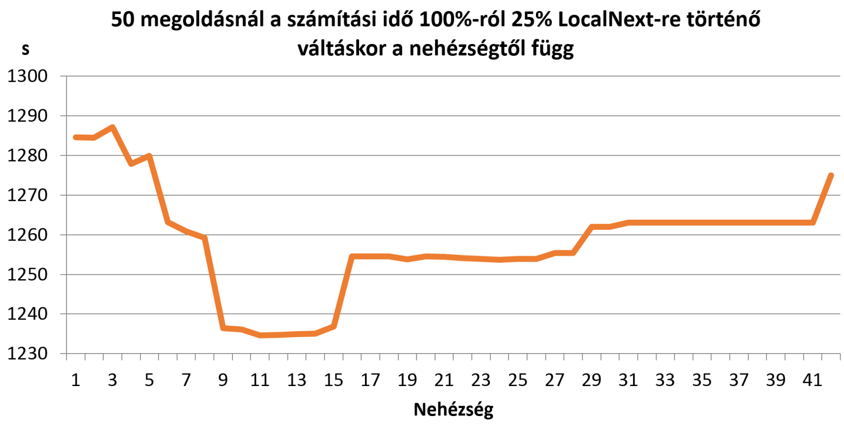
6.5. ábra. Két szálon futtatni csak akkor éri meg, ha a kért megoldások száma és a feladat mérete is kicsi

Megosztási stratégia: Egy másik fontos kérdés, hogy a már felfedezett keresőfának mely részéről kell részproblémát küldenie a szálnak a postaládába, ha egy másik várakozik. Ezt röviden megosztási stratégiának hívják. Az alsóbb szintekről való megosztást *"LocalNext"* stratégiának nevezik és felgyorsíthatja a keresést, valamint megoldást találhat egyfajta mohó stratégiát megvalósítva. Ezzel ellentétben, ha magasabb szintről történik a megosztás, azt *"GlobalNext"* stratégiának hívják, és csökkenti annak a kockázatát, hogy túl sok erőforrás legyen elpazarolva egyetlen irányba. A mérések kimutatták, ahogy az a 6.6. ábrán is látható, hogy a legtöbb esetben a *LocalNext* stratégia a jobb választás.

Ennek ellenére, ha a kért megoldásszám igen magas (50 feletti), a 25% LocalNext stratégia bizonyul a leghatékonyabbnak a bonyolultabb problémáknál, vagyis ahol már a nulladik lépésnél is látható, hogy legalább 10 műveleti egységről kell dönteni. Ez a 6.7. ábrán látható. Továbbá, ha



6.6. ábra. Keresési stratégiák összehasonlítása

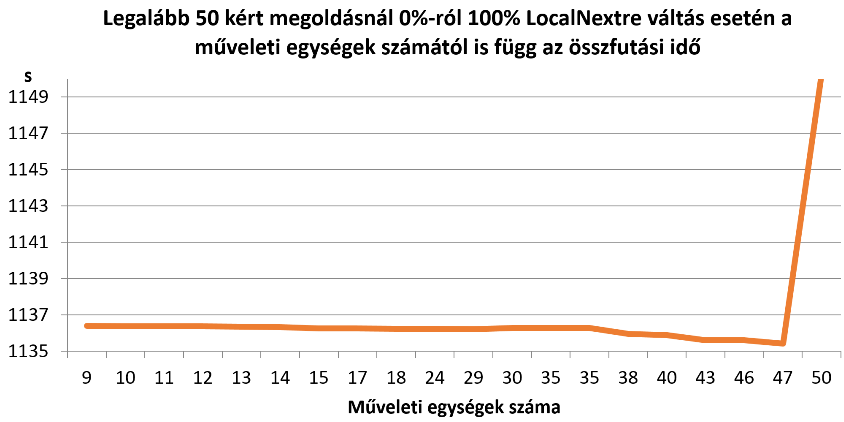


6.7. ábra. 25% LocalNext stratégiát szerencsés 10-es szintű nehézség fölött választani

a kért megoldások száma minimum 50, de legfeljebb 50 műveleti egységet tartalmaz a probléma, akkor a *GlobalNext* stratégia bizonyul a leggyorsabbnak. A 6.8. ábrán látható, hogy 50 kért megoldásnál 50 műveleti egység környékén szükséges stratégiát váltani. Ez a szabály erősebb, mint az előző.

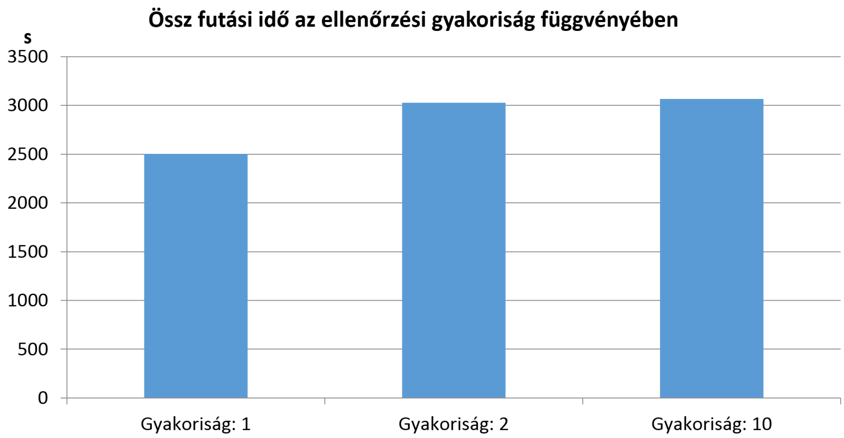
Az eredmények kimutatták, hogy a GlobalNext stratégia akkor hatékony, ha majdnem a teljes keresőfát be kell járni ahhoz, hogy a kért számú megoldást az algoritmus megtalálja. Ebben az esetben a párhuzamos szálak szinte függetlenül működhetnek, hiszen elegendő minimális kommunikáció is.

Ellenőrzési gyakoriság: Az ellenőrzési gyakoriság a párhuzamosított algoritmus harmadik paramétere. Ennek a beállítása azért fontos, mert amikor egy szál ellenőrzi a postaládát, akkor az összes többi szál nem fér ahhoz hozzá. Gyakori ellenőrzés esetén ez késedelmet okozhat a számításban, ha viszont ritkán történik vizsgálat, úgy a szálak az információhiány miatt lehet,



6.8. ábra. GlobalNext stratégiát alkalmazni abban az esetben a legjobb, ha a műveleti egységek száma nem jelentős, de a kért megoldások száma magas

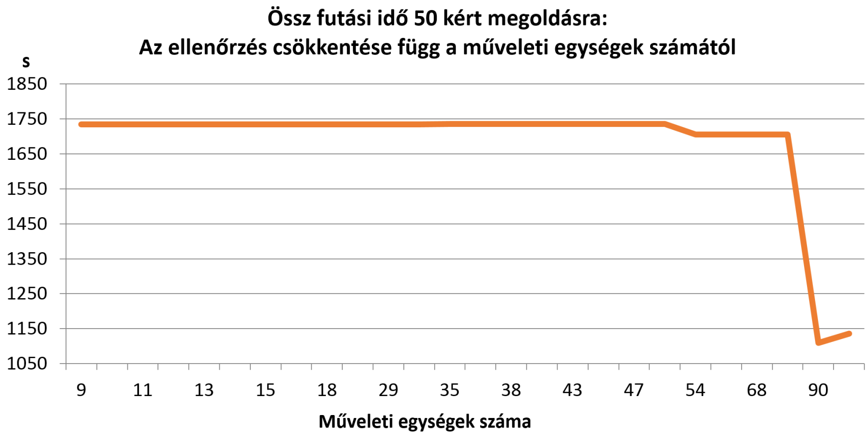
hogy felesleges számításokat végeznek. A számításokból kiderült, hogy annak ellenére, hogy az ellenőrzés blokkolja a postalását, mégis az a leghatékonyabb, ha az minden ciklusban ellenőrzésre kerül. Ez látható a 6.9. ábrán is.



6.9. ábra. Számítási idő különböző ellenőrzési gyakoriság mellett

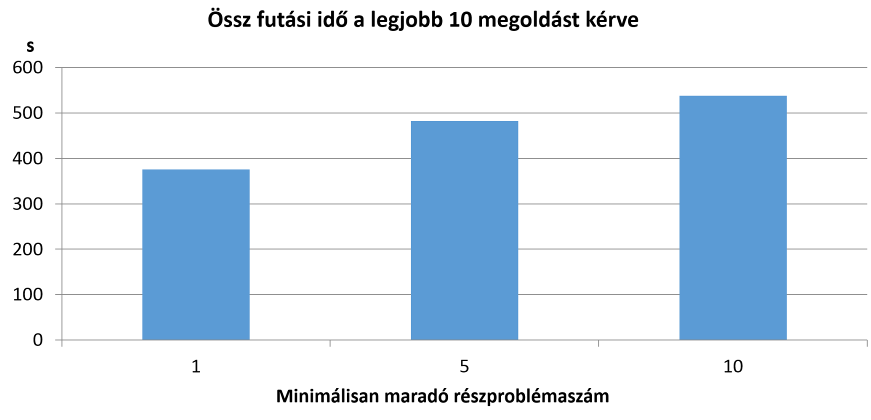
Amennyiben a kért megoldások száma legalább 50, és legalább 90 műveleti egységet tartalmaz a struktúra, úgy elegendő a postaládát csak minden második ciklusban ellenőrizni. Az 6.10. ábrán megfigyelhető, hogy a törés 50 kért megoldás esetén 90 műveleti egységnél van. A tapasztalatok azt mutatják, hogy általában ritkán ürül ki az egyes szálak saját részprobléma tárolója, viszont abban az esetben egy másik szál azonnal meg tud osztani velük.

Saját tárolóban maradó részproblémák minimum száma: Az utolsó paraméter ami befolyásolni tudja a megoldó működését a megosztáskor saját tárolóban maradó részproblémák



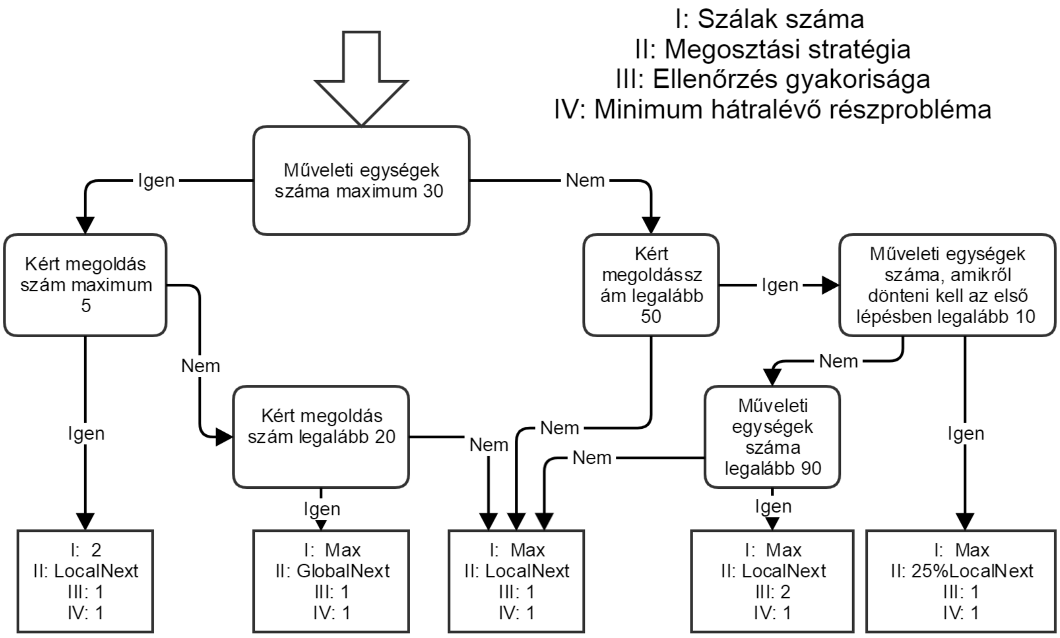
6.10. ábra. Ha több, mint 50 a kért megoldások száma, és a struktúra 90 vagy afölötti műveleti egységgel rendelkezik, hatékony, ha csak minden második ciklusban történik ellenőrzés

minimum száma, vagyis, hogy megosztás után legalább hány részprobléma kell, hogy maradjon a szál saját tárolójában ahhoz, hogy megoszthasson egyet a többiekkel. Ha ez az érték alacsony, úgy elképzelhető, hogy a szál gyakran kifogyhat a saját részproblémáiból és másoktól kell visszakérnie, ami felesleges körökhöz vezet. Ha ez a szám túl magas, a többi szálnak kellhet túl sokáig várnia ahhoz, hogy részproblémát kapjon amennyiben kiürült a saját tárolója. A tesztek kimutatták, hogy nem kell az előbbitől lényegesen tartani, és a szál akkor is megoszthat, ha utána csupán egy részprobléma marad a saját tárolójában, vagyis ha csak két részproblémával rendelkezik, abból is megoszthatja az egyiket (6.11. ábra).



6.11. ábra. A számítási idő függ attól is, hogy minimum hány részprobléma kell, hogy maradjon a szálak saját tárolójában megosztás után

A tesztek eredményét összegyűjtve elkészítettem a 6.12. ábrán látható folyamatábrát, ami a probléma struktúráját alapul véve segítséget nyújthat abban, hogy mi az optimális paraméterbeállítás. Ez a döntési folyamat a P-gráf megoldójában is implementálásra került. A párhuzamosított RCABB algoritmus indulása előtt a szoftver ellenőrizni a feladat struktúráját, vagyis a műveleti egységek számát, azt, hogy minimum hány műveleti egység van az első lépéskor amikről majd dönteni kell, illetve a kért megoldások számát, és ennek megfelelően a folyamatábra alapján elvégzi a megfelelő paraméterbeállítást.



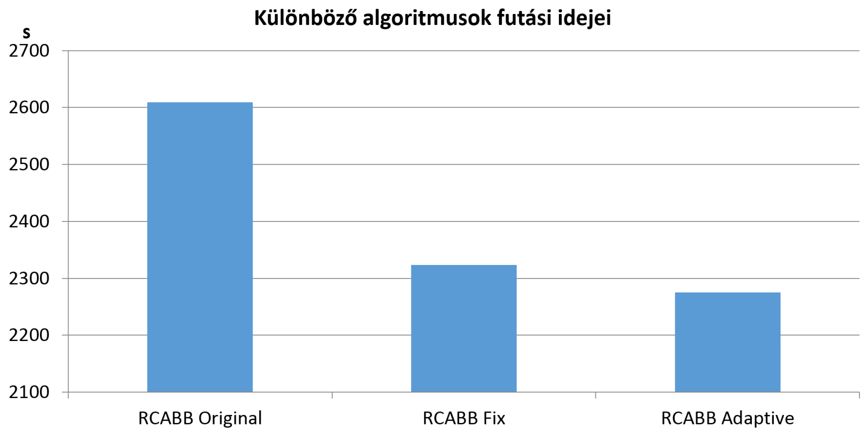
6.12. ábra. Folyamatábra a legjobb paraméterbeállítás kiválasztásához

6.4. Az optimalizált megoldó hatékonyságnövekedése és összehasonlítás más megoldókkal

A paraméteroptimalizálás után felmerült a kérdés, hogy mennyivel is sikerült növelni a hatékonyságát az algoritmusnak, ezért összehasonlításra került az adaptív, a fix legjobb és az eredeti paraméterbeállítás. (A nem párhuzamos megoldóhoz képest jelentős a hatékonyságnövekedés, ami az előző fejezetben bemutatásra is került, így itt azzal nem releváns összehasonlítani.) A 6.13. ábrán látható, hogy fixen az átlagosan legjobb paraméterbeállítások használata 11%-al növelte a párhuzamosított algoritmus hatékonyságát futási összidőre nézve, 1, 5, 10, 20 és 50 megoldásokat kérve a 6.2. fejezetben bemutatott teszthalmazon. Az átlagosan legjobb paraméterbeállítások az alábbiak:

– A lehető legtöbb szál használata a számításokhoz

– *LocalNext* megosztási stratégia alkalmazása

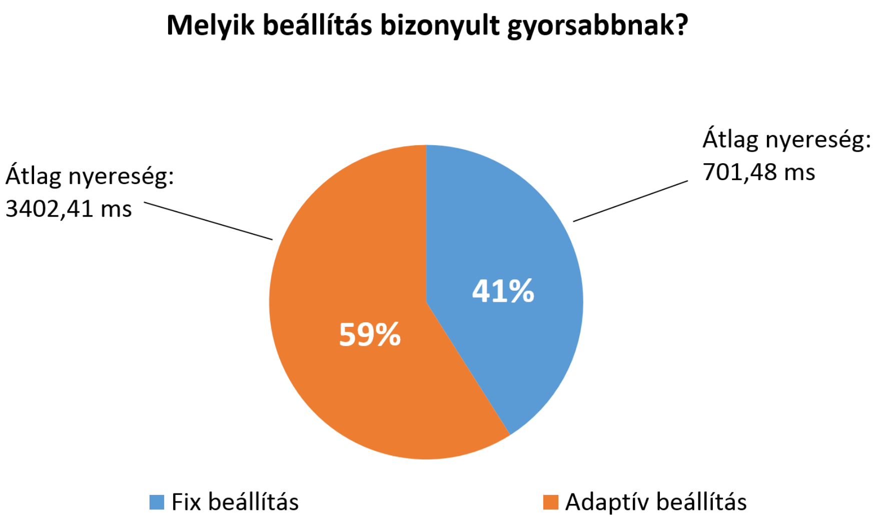


6.13. ábra. Az eredeti, a fix legjobb, valamint az adaptív paraméterbeállítással rendelkező RCABB futási idejének összehasonlítása

– A megosztási kérelmek ellenőrzése minden ciklusban

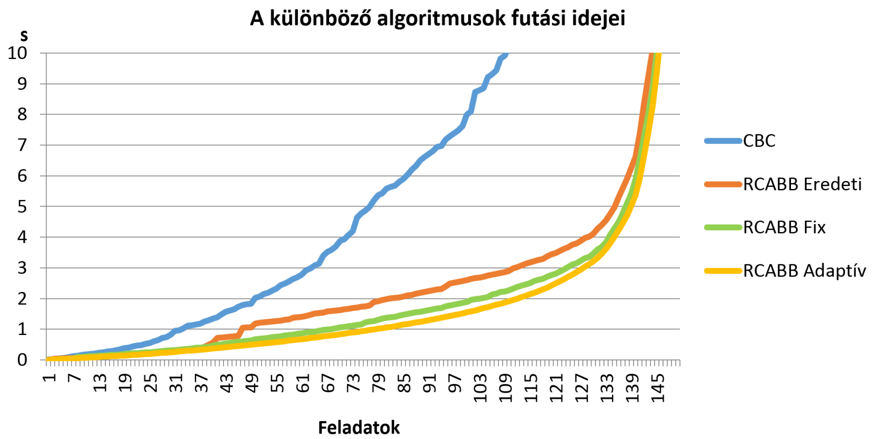
– Legalább egy, saját tárolóban maradó részprobléma a megosztás után

Még 2% teljesítménynövekedés érhető el a fix beállítások helyett adaptív beállítás használatával, vagyis amikor a szoftver a megoldás előtt a probléma struktúráját megvizsgálva maga állítja be a paramétereket. Fontos figyelembe venni, hogy a problémák mintegy harmadánál az adaptív beállítás akár 10%-al is képes csökkenteni a számítási időt, azonban meg kell jegyezni, hogy minél több a kért megoldások száma, annál kisebb lesz a különbség hatékonyság szempontjából a fix és az adaptív beállítások között.

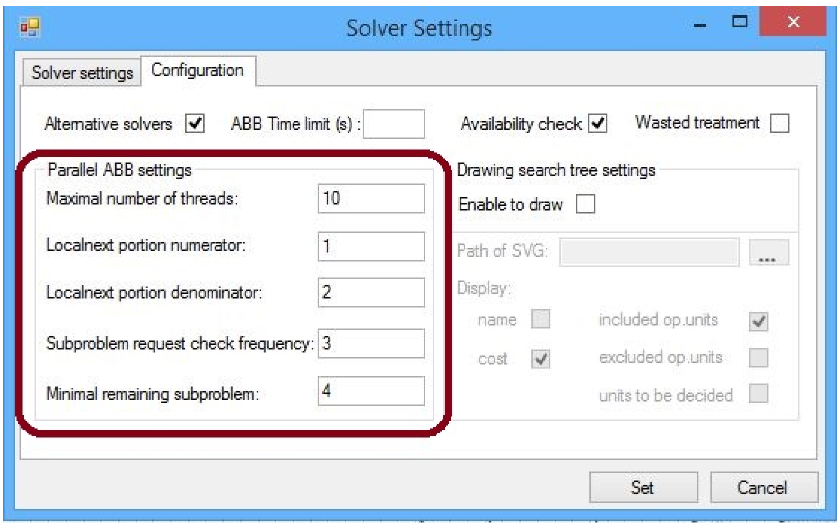


6.14. ábra. Az adaptív és fix beállítások összehasonlítása

A 6.14. ábráról leolvasható, hogy az esetek 59%-ában az adaptív megoldás bizonyult hatékonyabbnak, viszont ekkor a megoldó átlagosan 3,4 másodperccel volt gyorsabb, mint a fix beállítás mellett. Azokban az esetekben, amikor a fix beállítás bizonyult hatékonyabbnak, ez az előny csupán 0,7 másodperc volt, vagyis több időt lehet veszíteni ha az adaptív beállítás lenne a hatékonyabb, a választás mégis a fix beállításra esik. Kijelenthető tehát, hogy megéri adaptív beállításokkal használni a megoldót.



6.15. ábra. A COIN-OR CBC, a párhuzamosítatlan RCABB, a fix paraméterbeállítású RCABB valamint az adaptív paraméterbeállítású RCABB összehasonlítása



6.16. ábra. A párhuzamosított algoritmus manuális paraméterezhetősége a P-graph Studioban

Az elkészült párhuzamos, illetve paraméter-optimalizált megoldó szerencsés, ha nem csak saját magával, de legalább egy független MILP megoldóval is összehasonlításra kerül. A választás a széles körben ismert és szabadon felhasználható COIN-OR CBC-re esett, mivel ugyanaz a lineáris programozási megoldó az alapja, nevezetesen a COIN-OR CLP, ami az RCABB LP megoldójának is az alapja [77, 78]. A 6.15. ábrán látható összehasonlítás megmutatja, hogy a párhuzamosított RCABB algoritmus, majd a fix paraméterbeállítású, illetve az adaptív paraméterbeállítású RCABB algoritmus is sokkal hatékonyabb, mint a COIN-OR CBC a legjobb megoldás keresésében a már említett teszt halmazon mérve. Az ábrán a problémák futási idő szerint vannak rendezve. Fontos megjegyezni, hogy a COIN-OR CBC megoldó is párhuzamosított, és a tesztek során végzett CPU vizsgálat igazolta, hogy mind az RCABB, mind pedig a COIN-OR CBC a számításokhoz kihasználja az összes rendelkezésre álló szálat.

Az elkészített tesztfájlok és a mérési eredmények online publikálásra kerültek [80]. Az eredményes tesztelések után a párhuzamosított RCABB algoritmus integrálásra került a P-graph Studio megoldójába [10]. A megoldó alapértelmezetten az adaptív beállításokkal működik, de lehetőség van egyéni beállításokra is a "Preferences" - "Solver Settings" menüpont alatt. Ez látható a 6.16. ábrán is. A paraméterbeállítások csak akkor válnak elérhetővé, ha a szálak száma külön megadásra kerül.

6.5. A fejezet rövid összefoglalása

A fejezetben ismertettem a folyamathálózat-szintézis megoldó algoritmusának, az RCABB-nek a párhuzamosítását, ami a korábbi megoldással ellentétben már képes több processzormagon, elosztottan futtatni a Branch & Bound alapú algoritmust. A szálak közötti kommunikáció közös tárolókon alapszik, ahol a szálak részproblémákat tudnak megosztani egymással, illetve le tudják kérdezni az aktuális korlátot is, hiszen a megoldás-struktúrák tárolása is a közös részben valósul meg. Az elkészített megoldó a szálakat egyenletesen terheli. Az algoritmus jóságának méréséhez létrehoztam egy teszthalmaz bázist, ami nyilvánosan elérhető, így alapot képezhet a későbbi algoritmusfejlesztésekhez is. A párhuzamosítás megvalósítása több kérdést is felvet, melyek paraméterként jelennek meg. Ilyen lehet a futtatáshoz használt szálak száma vagy az ellenőrzési gyakoriság. A teszthalmaz segítségével elsőként megállapítottam, hogy általánosan mely beállítások bizonyulnak a leghatékonyabbnak, majd a probléma struktúrájától függően, adaptívan határoztam meg a legjobb beállítást. Az elért eredményeket más megoldókkal is összevetettem, ahol az általam elkészített algoritmus egyértelműen hatékonyabbnak bizonyult. Ezt követően integráltam a párhuzamosított, adaptív paraméterbeállítású megoldót a P-graph Studioba, meghagyva a lehetőséget a manuális beállításnak.

6.5.1. A fejezethez tartozó tézis

Elkészítettem és optimalizáltam a P-gráf egyik megoldó algoritmusának párhuzamosított változatát.

(a) Kidolgoztam a P-gráfhoz tartozó RCABB algoritmus párhuzamosítását, ami garantálja az N-legjobb megoldást is.

(b) Létrehoztam egy mindenki számára elérhető teszt-bázist a PNS megoldók teszteléséhez.

(c) A létrehozott tesztbázis segítségével meghatároztam, hogy általánosságban mely paraméterbeállítások a legjobbak az elkészült párhuzamosított algoritmushoz.

(d) Feladatosztályokat hoztam létre, és ezen osztályokra külön-külön határoztam meg a legjobb paraméterbeállítást. Az osztályozás segítségével az algoritmus paraméterbeállítását adaptívvá tettem.

(e) Kísérletileg igazoltam az elkészült megoldó hatékonyságát, más, ingyenesen elérhető megoldók összehasonlításán keresztül.

6.5.2. A fejezet témaköréhez kapcsolódó publikációk

Nemzetközi folyóiratcikk

- Bartos Anikó és Bertók Botond: Parameter tuning for a cooperative parallel implementation of processnetwork synthesis algorithms, folyóirat: Central European Journal of Operations Research, 1-22. oldal, 2018. (IF = 1,26) [29]

Nemzetközi konferencia-kiadványokban megjelent közlemények

- Bartos Anikó és Bertók Botond: Analysis of Search Strategies for Parallel Implementation of a Process-Network Synthesis Solver, kiadvány: ASCONIKK 2014, 13. oldal (2014) [81]
- Bartos Anikó és Bertók Botond: Synchronization and Load Distribution Strategies for Parallel Implementations of P-graph Optimizer, kiadvány: MACRo 2015, 303-313. oldal (2015) [82]

Nemzetközi konferencia előadások

- Bartos Anikó: Teaching Tools in the Logistics Tasks, konferencia: TIIKM'S 1st Annual International Conference on Education, Peking, Kína, 2015. [83]
- Bartos Anikó és Bertók Botond: Parameter tuning for a cooperative parallel implementation of process-network synthesis algorithms, kiadvány: VOCAL Optimization Conference: Advanced Algorithms 2016, 96. oldal, 2016. [84]
- Bartos Anikó és Bertók Botond: Energy Storage Capacity Optimization for Residential Areas, konferencia: 2nd eseia Conference on Smart and Green Transitions in Cities and Regions, Graz, Ausztria, 2016. [85]

7. fejezet

Összefoglalás

A dolgozatban bemutatásra kerültek a folyamathálózat-szintézishez szorosan kapcsolódó új, tudományos eredmények. A 4. fejezet átfogóan foglalkozott a multiperiódusos modellekkel. A korábban publikált eredményekkel ellentétben létrehozásra került egy általános modell-leírás a több periódusból álló feladatokhoz, illetve bevezetésre került a periódusok közötti tárolás megvalósítása. Az általános modellek alapján a P-graph Studio szoftverben implementálásra is kerültek ezen új részek. A fejezet részletesen is bemutat több, ehhez a témához kapcsolódó esettanulmányt, és azok P-gráffal történő megoldását. Ezek közül is kiemelkedik az energia tárolása, és az áram szolgáltatás ütemezése microgrid rendszerben. Az eredmények egyértelműen igazolják a módszer hatékonyságát.

Az 5. fejezet egy ütemezési problémával, a gyártósor kiegyensúlyozással foglalkozik. A probléma egy korábbi, a témában publikált modell alapján megvalósítható, viszont szigorú korlátai miatt annál általánosabban is leírható, így könnyen automatizálhatóvá válik. Az általános modell leíráson túl a fejezet bemutatja a kapcsolódó, elkészült szoftvert, ami valós környezetben, egy összeszerelő üzemben is tesztelésre került. A visszajelzések megmutatták, hogy a korábbi kiosztáshoz képest a modellel és szoftverrel támogatott sorkiosztással akár 20-25%-nyi hatékonyságnövekedés is elérhető.

Az utolsó, 6. fejezet a P-gráf egyik alap algoritmusának, az RCABB-nek a párhuzamosításáról és annak paraméteroptimalizálásáról szól. Az ehhez kapcsolódó kutatások egyik "melléktermékeként" létrehozásra került egy mindenki számára elérhető tesztadathalmaz. A párhuzamosítással és a megfelelő paraméterbeállítással jelentősen gyorsult az algoritmus futása. Az optimalizált algoritmus az ingyenesen elérhető P-graph Studio alatt is implementálásra került.

Összességében kijelenthető, hogy az elért eredmények újak, és önálló kutatás részeként kerültek megvalósításra. A létrejött algoritmusok a gyakorlatban használt szoftverek alatt is kivitelezésre kerültek, a visszajelzések pedig pozitívak voltak. Elmondható tehát, hogy nem csak elméleti, de gyakorlatban is hasznos eredmények születtek.

Irodalomjegyzék

[1] E Kondili, CC Pantelides, and RWH Sargent. "A general algorithm for short-term scheduling of batch operations—I. MILP formulation". In: *Computers & Chemical Engineering* 17.2 (1993), pp. 211–227.

[2] CG Cassandras and S Lafortune. *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[3] B Berthomieu and M Menasche. "An enumerative approach for analyzing time Petri nets". In: *Proceedings IFIP*. Citeseer. 1983.

[4] E Sanmarti, F Friedler, and L Puigjaner. "Combinatorial technique for short term scheduling of multipurpose batch plants based on schedule-graph representation". In: *Computers & chemical engineering* 22 (1998), S847–S850.

[5] E Sanmarti et al. "Combinatorial framework for effective scheduling of multipurpose batch plants". In: *AIChE Journal* 48.11 (2002), pp. 2557–2570.

[6] F Friedler et al. "Combinatorial algorithms for process synthesis". In: *Computers & Chemical Engineering* 16 (1992), pp. 313–320. DOI: 10.1016/S0098-1354(09)80037-9.

[7] JJ Siirola. "Industrial applications of chemical process synthesis". In: *Advances in chemical engineering*. Vol. 23. Elsevier, 1996, pp. 1–62.

[8] B Bertok, M Barany, and F Friedler. "Generating and analyzing mathematical programming models of conceptual process design by P-graph software". In: *Industrial & Engineering Chemistry Research* 52 (2013), pp. 166–171. DOI: 10.1021/ie301155n.

[9] F. Friedler et al. "Combinatorially accelerated branch-and-bound method for solving the MIP model of process network synthesis". In: *State of the Art in Global Optimization* (1996), pp. 609–626. DOI: 10.1007/978-1-4613-3437-8_35.

[10] Uni-Pannon:DCS. *P-Graph Studio*. Version 5.2.2.2. 2018. URL: http://p-graph.org/.

[11] A Bartos and B Bertok. "Production line balancing by P-graphs". In: *Optimization and Engineering* (2019), pp. 1–18. DOI: 10.1007/s11081-019-09462-1.

[12] E Konig, Z Sule, and B Bertok. "Comparison of optimization techniques in the P-graph framework for the design of supply chains under uncertainties". In: *INFORMATION SECURITY* (), p. 35.

[13] F Friedler, JB Varga, and LT Fan. "Decision-mapping: a tool for consistent and complete decisions in process synthesis". In: *Chemical Engineering Science* 50.11 (1995), pp. 1755–1768.

[14] LT Fan et al. "Mechanisms of ammonia-synthesis reaction revisited with the aid of a novel graph-theoretic method of determining candidate mechanisms in deriving the rate law of a catalytic reaction". In: *Hungarian Journal of Industrial Chemistry* 29.1 (2001), pp. 71–80.

[15] F Friedler et al. "Graph-theoretic approach to process synthesis: polynomial algorithm for maximal structure generation". In: *Computers & Chemical Engineering* 17 (1993), pp. 929–942. DOI: 10.1016/0098-1354(93)80074-W.

[16] Z Kovacs et al. "Separation-network synthesis: global optimum through rigorous superstructure". In: *Computers & Chemical Engineering* 24.8 (2000), pp. 1881–1900.

[17] J Varga. "Extensions of the Process-Network Synthesis Problem". PhD thesis. Univesity of Veszprem, 2000.

[18] AB Nagy et al. "Integrated synthesis of process and heat exchanger networks: algorithmic approach". In: *Applied Thermal Engineering* 21.13-14 (2001), pp. 1407–1427.

[19] LT Fan, B Bertok, and F Friedler. "A graph-theoretic method to identify candidate mechanisms for deriving the rate law of a catalytic reaction". In: *Computers & chemistry* 26.3 (2002), pp. 265–292.

[20] MS Peters et al. *Plant design and economics for chemical engineers*. Vol. 4. McGraw-Hill New York, 1968.

[21] Sz Gyapay and A Pataricza. "A combination of Petri nets and Process Network Synthesis". In: *SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme-System Security and Assurance (Cat. No. 03CH37483)*. Vol. 2. IEEE. 2003, pp. 1167–1174.

[22] I Heckl, F Friedler, and LT Fan. "Solution of separation-network synthesis problems by the P-graph methodology". In: *Computers & Chemical Engineering* 34.5 (2010), pp. 700–706.

[23] M Barany et al. "Solving vehicle assignment problems by process-network synthesis to minimize cost and environmental impact of transportation". In: *Clean Technologies and Environmental Policy* 13.4 (2011), pp. 637–642. DOI: 10.1007/s10098-011-0348-2.

[24] K Kalauza et al. "Extending process-network synthesis algorithms with time bounds for supply network design". In: *Chemical Engineering* 29 (2012). DOI: 10.3303/CET1229044.

[25] JC Garcia-Ojeda, B Bertok, and F Friedler. "Planning evacuation routes with the P-graph framework". In: *Chemical Engineering Transactions* 29 (2012), pp. 1531–1536. DOI: 10.3303/CET1229256.

[26] B Bertok et al. "Combinatorial algorithm for synthesizing redundant structures to increase reliability of supply chains: application to biodiesel supply". In: *Industrial & Engineering Chemistry Research* 52.1 (2012), pp. 181–186. DOI: 10.1021/ie301393d.

[27] M Frits and B Bertok. "Process scheduling by synthesizing time constrained process-networks". In: *Computer Aided Chemical Engineering*. Vol. 33. Elsevier, 2014, pp. 1345–1350. DOI: 10.1016/B978-0-444-63455-9.50059-3.

[28] I Heckl et al. "Process synthesis involving multi-period operations by the P-graph framework". In: *Computers & Chemical Engineering* 83 (2015), pp. 157–164. DOI: 10.1016/j.compchemeng.2015.04.037.

[29] Aniko Bartos and Botond Bertok. "Parameter tuning for a cooperative parallel implementation of ProcessNetwork Synthesis algorithms". In: *Central European Journal of Operations Research* (2018), pp. 1–22.

[30] KB Aviso, JY Lee, and RR Tan. "A P-graph model for multi-period optimization of isolated energy systems". In: *Chemical Engineering Transactions* 52 (2016), pp. 865–870. DOI: 10.3303/CET1652145.

[31] PS Varbanov, F. Friedler, and JJ Klemes. "Process network design and optimisation using P-graph: The success, the challenges and potential roadmap". In: *Chemical Engineering Transactions* 61 (2017), pp. 1549–1554. DOI: 10.3303/CET1761256.

[32] TG Walmsley et al. "Total site utility system structural design using P-graph". In: *Chemical Engineering Transactions* 63 (2018), pp. 31–36. DOI: 10.3303/CET1863006.

[33] B Bertok and A Bartos. "Algorithmic process synthesis and optimisation for multiple time periods including waste treatment: Latest developments in P-graph Studio software". In: *Chemical Engineering Transactions* 70 (2018), pp. 97–102.

[34] I Heckl et al. "Modeling multi-period operations using the P-graph methodology". In: *Computer Aided Chemical Engineering* 33 (2014), pp. 979–984. DOI: 10.1016/B978-0-444-63456-6.50164-2.

[35] A Keresszegi. "PNS Draw". In: (2010).

[36] Uni-Pannon:DCS. "PNS Studio". In: (2011).

[37] T Koch. "ZIMPL". In: (2001).

[38] T Koch. "Rapid mathematical programming". In: (2005).

[39] I Chakroun and N Melab. "Operator-level gpu-accelerated branch and bound algorithms". In: *Procedia Computer Science* 18 (2013), pp. 280–289.

[40] Y Evtushenko, M Posypkin, and I Sigal. "A framework for parallel large-scale global optimization". In: *Computer Science-Research and Development* 23.3-4 (2009), pp. 211–215.

[41] B Bourbeau, TG Crainic, and B Gendron. "Branch-and-bound parallelization strategies applied to a depot location and container fleet management problem". In: *Parallel Computing* 26.1 (2000), pp. 27–46.

[42] J Clausen and M Perregaard. "On the best search strategy in parallel branch-and-bound: Best-First Search versus Lazy Depth-First Search". In: *Annals of Operations Research* 90 (1999), pp. 1–17.

[43] EA Pruul, GL Nemhauser, and RA Rushmeier. “Branch-and-bound and parallel computation: A historical note”. In: *Operations Research Letters* 7.2 (1988), pp. 65–69.

[44] DA Bader, WE Hart, and CA Phillips. “Parallel algorithm design for branch and bound”. In: *Tutorials on Emerging Methodologies and Applications in Operations Research*. Springer, 2005, pp. 5–1.

[45] U Honig and W Schiffmann. “A parallel branch-and-bound algorithm for computing optimal task graph schedules”. In: *International conference on grid and cooperative computing*. Springer. 2003, pp. 18–25.

[46] C Cartis, JM Fowkes, and NIM Gould. “Branching and bounding improvements for global optimization algorithms with Lipschitz continuity properties”. In: *Journal of Global Optimization* 61.3 (2015), pp. 429–457.

[47] L Barreto and M Bauer. “Parallel branch and bound algorithm-a comparison between serial, openmp and mpi implementations”. In: *Journal of Physics: Conference Series* 256.1 (2010), p. 012018.

[48] J Eckstein. “Distributed versus centralized storage and control for parallel branch and bound: mixed integer programming on the CM-5”. In: *Computational Optimization and Applications* 7.2 (1997), pp. 199–220.

[49] GH Dastghaibifard et al. “A parallel branch and bound algorithm for vehicle routing problem”. In: *Proceedings of the International MultiConference of Engineers and Computer Scientists*. Vol. 2. 2008, pp. 19–21.

[50] TG Crainic, B Le Cun, and C Roucairol. “Parallel branch-and-bound algorithms”. In: *Parallel combinatorial optimization* 1 (2006), pp. 1–28.

[51] DL Miller and JF Pekny. “Results from a parallel branch and bound algorithm for the asymmetric traveling salesman problem”. In: *Operations Research Letters* 8.3 (1989), pp. 129–135.

[52] M Mezmaz, N Melab, and D Tuyttens. “A multithreaded branch-and-bound algorithm for solving the flow-shop problem on a multicore environment”. In: *Large Scale Network-Centric Distributed Systems* (2013), pp. 53–70.

[53] JB Varga, F Friedler, and LT Fan. “Parallelization of the accelerated branch-and-bound algorithm of process synthesis: application in total flowsheet synthesis”. In: *Acta Chimica Slovenica* 42 (1995), pp. 15–15.

[54] RW McPherson, Ch M Starks, and GJ Fryar. “Vinyl chloride monomer... What you should know”. In: *Chemischer Informationsdienst* 10.28 (1979), no–no.

[55] A Lakshmanan and LT Biegler. “A case study for reactor network synthesis: the vinyl chloride process”. In: *Computers & Chemical Engineering* 21 (1997), S785–S790.

[56] JA Cowfer and AJ Magistro. *Vinyl Polymers, Vinyl Chloride. Volume 23 of (Eds: Martin Grayson and David Eckroth) Kirk-Othmer Encyclopedia of Chemical Technology*. 1983.

[57] H Chen et al. "Autonomous demand side management based on energy consumption scheduling and instantaneous load billing: An aggregative game approach". In: *IEEE Transactions on Smart Grid* 5.4 (2014), pp. 1744–1754. DOI: 10.1109/TSG.2014.2311122.

[58] CPC Bong et al. "The role of smart waste management in smart agriculture". In: *Chemical Engineering Transactions* 70 (2018), pp. 937–942. DOI: 10.3303/CET1870157.

[59] WH Liu et al. "Power Pinch Analysis supply side management: strategy on purchasing and selling of electricity". In: *Clean Technologies and Environmental Policy* 18.8 (2016), pp. 2401–2418. DOI: 10.1007/s10098-016-1213-0.

[60] HM Liou. "The development of electricity grid, smart grid and renewable energy in Taiwan". In: *Smart Grid and Renewable Energy* 8.06 (2017), p. 163. DOI: 10.4236/sgre.2017.86011.

[61] State Government of Victoria. *Smart meters executive summary*. 2017. URL: https://www.energy.vic.gov.au/electricity/smart-meters/reports-and-consultations/advanced-metering-infrastructure-customer-impacts-study-volume-1/executive-summary (visited on 04/19/2019).

[62] UCI Machine Learning Repository. *Individual household electric power consumption Data Set*. 2012. URL: https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption (visited on 04/19/2019).

[63] E Konig and B Bertok. "Process graph approach for two-stage decision making: Transportation contracts". In: *Computers & Chemical Engineering* 121 (2019), pp. 1–11. DOI: 10.1016/j.compchemeng.2018.07.011.

[64] B Bertok and A Bartos. "Renewable energy storage and distribution scheduling for microgrids by exploiting recent developments in Process Network Synthesis". In: *Journal of Cleaner Production* (2020).

[65] A Bartos, B Bertok, and A Szlama. "Optimal design of multi-period process networks including storages for renewable resources". In: *International Congress on Sustainability Science Engineering ICOSSE*. 2015.

[66] A Bartos and B Bertok. "P-graph framework: Computer aided model generation and solution for supply network optimization problems". In: *European Working Group on Location Analysis Meeting 2015*. 2015, p. 29.

[67] E Konig, A Bartos, and B Bertok. "Free software for the education of supply chain optimization". In: *VOCAL Optimization Conference: Advanced Algorithms 2016*. 2016, p. 17.

[68] A Bartos and B Bertok. "Estimation of the return of investment in new technologies regarding periodically changing demands, availability of resources, and storages". In: *Chemical Engineering Days 2017*. 2017.

[69] A Bartos and B Bertok. *Software for economical evaluation of utilizing periodically available renewable resources*. 2017.

[70] P Sivasankaran and P Shahabudeen. “Literature review of assembly line balancing problems”. In: *The International Journal of Advanced Manufacturing Technology* 73.9-12 (2014), pp. 1665–1694. DOI: 10.1007/s00170-014-5944-y.

[71] JM Wilson. “Henry Ford vs. assembly line balancing”. In: *International Journal of Production Research* 52.3 (2014), pp. 757–765. DOI: 10.1080/00207543.2013.836616.

[72] ME Salveson. “The assembly line balancing problem”. In: *The Journal of Industrial Engineering* (1955), pp. 18–25.

[73] N Kumar and D Mahto. “Assembly line balancing: a review of developments and trends in approach to industrial application”. In: *Global Journal of Research In Engineering* (2013).

[74] MRA Make, MFFA Rashid, and MM Razali. “A review of two-sided assembly line balancing problem”. In: *The International Journal of Advanced Manufacturing Technology* 89.5-8 (2017), pp. 1743–1763. DOI: 10.1007/s00170-016-9158-3.

[75] M Aghajani, R Ghodsi, and B Javadi. “Balancing of robotic mixed-model two-sided assembly line with robot setup times”. In: *The International Journal of Advanced Manufacturing Technology* 74.5-8 (2014), pp. 1005–1016. DOI: 10.1007/s00170-014-5945-x.

[76] H Fazlollahtabar, H Hajmohammadi, and A Es’haghzadeh. “A heuristic methodology for assembly line balancing considering stochastic time and validity testing”. In: *The International Journal of Advanced Manufacturing Technology* 52.1-4 (2011), pp. 311–320. DOI: 10.1007/s00170-010-2708-1.

[77] J Forrest et al. “coin-or/Cbc: Version 2.9.9”. In: 1317566 (2018). DOI: 10.5281/zenodo.

[78] J Forrest and R Lougee-Heimer. “CBC user guide”. In: *Emerging theory, methods, and applications*. INFORMS, 2005, pp. 257–277. DOI: 10.1287/educ.1053.0020.

[79] B Bertok, F Friedler, and LT Fan. “Random generation of test problems for process synthesis”. In: *CHISA ’98 (13th International Congress of Chemical and Process Engineering)* (1998).

[80] *Test base*. p-graph.org/wp-content/uploads/2017/12/test_base.zip. Accessed: 2018-07-01.

[81] A Bartos and B Bertok. “Analysis of search strategies for parallel implementation of a Process-Network Synthesis solver”. In: *ASCONIKK 2014* (2014), p. 13.

[82] A Bartos and B Bertok. “Synchronization and load distribution strategies for parallel implementations of P-graph optimizer”. In: *MACRo 2015* 1.1 (2015), pp. 303–313.

[83] A Bartos. “Teaching tools in the logistics tasks”. In: *TIIKM’S 1st Annual International Conference on Education*. Vol. 1. 2015, p. 27.

[84] A Bartos and B Bertok. “Parameter tuning for a cooperative parallel implementation of Process-Network Synthesis algorithms”. In: *VOCAL Optimization Conference: Advanced Algorithms 2016*. 2016, p. 96.

[85] A Bartos and B Bertok. “Energy storage capacity optimization for residential areas”. In: *2nd eseia Conference on Smart and Green Transitions in Cities and Regions*. 2016, p. 1.

Melléklet

A "p-graph.org"-on található tesztadatok tulajdonágai:

Name	# of op.units	# of materials	difficulty	Name	# of op.units	# of materials	difficulty	Name	# of op.units	# of materials	difficulty	Name	# of op.units	# of materials	difficulty
0	9	14	6	39	15	26	0	78	35	57	3	117	70	117	0
1	10	20	4	40	15	25	0	79	38	65	17	118	70	120	13
2	10	25	6	41	15	29	0	80	38	55	15	119	70	111	2
3	10	18	4	42	15	22	0	81	40	55	14	120	70	108	15
4	10	15	5	43	15	23	0	82	43	74	7	121	70	115	11
5	10	14	6	44	15	26	0	83	46	67	9	122	70	115	14
6	11	14	4	45	15	25	0	84	46	68	0	123	70	123	0
7	11	21	3	46	15	23	0	85	47	77	20	124	70	104	19
8	11	17	1	47	15	33	6	86	50	78	0	125	70	113	18
9	11	18	7	48	15	21	8	87	50	84	8	126	70	112	30
10	12	23	7	49	17	28	1	88	50	93	15	127	70	108	20
11	12	20	0	50	18	27	7	89	50	81	2	128	70	117	18
12	13	16	5	51	18	31	0	90	50	84	15	129	70	107	19
13	13	22	0	52	18	30	12	91	50	80	7	130	70	113	6
14	13	26	2	53	24	48	9	92	50	79	9	131	70	115	4
15	13	20	6	54	24	37	1	93	50	90	8	132	70	109	4
16	14	22	7	55	29	46	0	94	50	81	10	133	70	115	14
17	14	25	0	56	29	52	0	95	50	87	21	134	70	101	26
18	14	28	7	57	30	57	9	96	50	78	5	135	70	102	7
19	14	14	6	58	30	55	5	97	50	83	19	136	90	140	5
20	15	27	3	59	30	49	9	98	50	87	13	137	90	158	13
21	15	28	10	60	30	50	10	99	50	73	12	138	90	143	3
22	15	26	10	61	30	53	11	100	50	81	7	139	90	160	26
23	15	24	7	62	30	53	4	101	50	95	14	140	90	133	4
24	15	31	3	63	30	48	16	102	50	85	8	141	90	148	3
25	15	23	8	64	30	48	3	103	50	80	9	142	90	140	1
26	15	24	6	65	30	49	8	104	50	82	0	143	90	155	41
27	15	27	8	66	30	54	19	105	50	86	12	144	90	142	23
28	15	31	5	67	30	53	12	106	50	101	0	145	90	149	28
29	15	31	4	68	30	53	10	107	50	85	0	146	90	136	4
30	15	22	2	69	30	58	13	108	50	80	8	147	90	143	2
31	15	26	6	70	30	50	0	109	50	90	0	148	90	149	3
32	15	22	5	71	30	51	0	110	50	86	0	149	90	143	8
33	15	33	5	72	30	40	0	111	50	68	5				
34	15	29	5	73	30	44	13	112	54	74	25				
35	15	29	7	74	30	41	0	113	55	97	3				
36	15	24	7	75	30	48	0	114	68	102	5				
37	15	30	3	76	30	50	0	115	70	101	30				
38	15	27	0	77	35	56	3	116	70	109	24				