MISKOLCI EGYETEM

GÉPÉSZMÉRNÖKI ÉS INFORMATIKAI KAR

HATVANY JÓZSEF INFORMATIKAI TUDOMÁNYOK DOKTORI ISKOLA

Modellek és módszerek többcélú, többprojektes és erőforráskorlátos projektütemezési feladatok megoldására

PhD értekezés

Készítette

Mihály Krisztián

Okleveles mérnök-informatikus

Témavezető:

Dr. Kulcsár Gyula

Egyetemi docens

A doktori iskola vezetője:

Prof. Dr. Szigeti Jenő

Miskolc, 2025

Declaration

The author hereby declares that this thesis has not been submitted, either in the same or in different form, to this or to any other university for obtaining Ph.D. degree. The author confirms that the submitted work is his own and the appropriate credit has been given where reference has been made to the work of others.

Nyilatkozat

Alulírott Mihály Krisztián kijelentem, hogy ezt a doktori értekezést magam készítettem, és abban csak a megadott forrásokat használtam fel. Minden olyan részt, amelyet szó szerinti vagy azonos tartalomban, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Miskolc, 2025.01.06.

Mihály Krisztián

Tartalomjegyzék

Összefoglalás

A dolgozatomban összefoglalom a többprojektes, többcélú, erőforráskorlátos projekt ütemezési feladatok modellezésével és megoldásával kapcsolatban végzett kutatómunkám elért eredményeit. Tevékenységem az erőforráskorlátos, projektütemezési feladatok modellezési kiterjesztésére fókuszált, melyek közül az egyik kiterjesztés a több projektet egyszerre kezelő irányra vonatkozik, miközben a másik kiterjesztésem a projektek egyedi célfüggvényeinek és a teljes rendszer célfüggvényeinek együttes kezelését támogatja.

Dolgozatom első fejezetében ismertetem a kutatómunkám témakörét, a kitűzött célokat és az alkalmazott kutatási eljárásokat és módszereket. A második fejezetben áttekintést adok a tématerülethez kapcsolódó, az irodalomban elérhető ismeretanyagról és aktuális kutatási eredményekről.

A harmadik fejezetben bemutatom a kutatási feladatom során megfogalmazott kiterjesztett erőforráskorlátos projektütemezési probléma matematikai modelljét. A matematikai leírás kiterjed a leírásban használt alapfogalmakra és azok jelölésére. Az alapfogalmak esetén példát mutatok a modell egyes gyakorlati alkalmazására. Bevezetésre kerülnek a probléma alapadataiból származtatott másodlagos bemeneti adatok. Meghatározom azokat a feltételeket és korlátokat, amelyeket a probléma leíró adatoknak teljesíteniük kell. Ezt követően meghatározásra kerülnek a probléma megoldására javasolt elsődleges döntési változók, amelyek a problémára adott megoldást reprezentálják. Az elsődleges döntési változók mellett a megértést segítő másodlagos minősítő értékek is bevezetésre kerülnek. Hasonlóan meghatározásra kerülnek azok a korlátok, amelyeket a döntési változóknak és a másodlagos minősítő értékeknek teljesíteniük kell, hogy a megoldást végrehajtható megoldásnak tekintsük. Végül az alkalmazható célfüggvényekre mutatok néhány példát.

A negyedik fejezetben bemutatok egy megoldási koncepciót a harmadik fejezetben ismertetett probléma megoldására. A koncepció épít a korábbi tudományos eredményekre és azok eredményeiből kiindulva teszek kiterjesztési javaslatokat. A megoldó eljárás két fő egységből áll, (1) egy kiterjesztett, rugalmasan vezérelhető ütemterv felépítő eljárásból, (2) egy metaheurisztikus keresőből, amely a felépítő eljárást beépülő módszerként (konstruktív szimulátorként) tudja irányítani és alkalmazni. A bemutatás során ismertetek egy kiterjesztett evolúciós algoritmust, amely bizonyos gyártásütemezési feladatok esetében a fejlesztés kiindulási alapját adó eredeti eljáráshoz viszonyítva hatékonyabb megoldásokat ad.

Az ötödik fejezetben a megoldási koncepció megvalósíthatósága érdekében végzett szoftverarchitektúra-tervezési és alkalmazásfejlesztési munkám eredményeit foglalom össze. A fejezet célja a megvalósítás koncepcionális ismertetése, amely kiterjed a kialakított szoftver kompozíciós felépítésére és egy javasolt objektum-orientált implementációjára.

A hatodik fejezetben bemutatom a modell alkalmazhatóságát különböző ismert gyártásütemezési és benchmark feladatokon. A bemutatás során az alkalmazott mérési módszert, futási eredményeket és a futási eredményekből levont következtetéseket ismertetem. A többprojektes kiterjesztés tesztelésére ismert benchmark feladatokból kiindulva készítettem

teszt adathalmazt és ezeken a feladatokon vizsgáltam a megoldó módszer rugalmasságát és hatékonyságát, miközben különböző célfüggvény konfigurációk szerint végeztem teszteléseket.

Az értekezés hetedik fejezete az új tudományos eredményeket tézisek formájában foglalja össze. Az értekezés további fejezeteiben közlöm a saját tudományos publikációs eredményeimet, a hivatkozott irodalmak listáját, valamint az értekezésben szereplő ábrák és táblázatok jegyzékét. A függelék a dolgozatban kiemelt futási eredményekkel kapcsolatos további részleteket tartalmaznak.

Summary

In my dissertation, I summarize the results of my research related to the modeling and solution of multi-project, multi-objective, resource-constrained project scheduling problems. My work focused on extending the modeling of resource-constrained project scheduling problems. One of extensions addresses the simultaneous handling of multiple projects, while the other extension supports the joint management of the individual objective functions of the projects and the overall system objective functions.

In the first chapter of my dissertation, I present the topic and the main goals of my research, the applied research procedures and methods. In the second chapter, I provide an overview of the knowledge available in the literature related to the subject area and current research contributions.

In the third chapter, I present the mathematical model of the extended resource-constrained project scheduling problem formulated during my research. The mathematical description covers the basic concepts, and their notations used in the model. I provide examples of practical applications of these concepts in the model. Secondary input data derived from the basic problem data are introduced. I define the conditions and constraints that the descriptive data of the problem must satisfy. Then, I specify the primary decision variables proposed to represent the solution to the problem. In addition to the primary decision variables, secondary auxiliary values are also introduced to facilitate understanding. Similarly, I define the constraints that both the decision variables and the secondary auxiliary values must meet for the solution to be considered feasible. Finally, I provide examples of possible objective functions.

In the fourth chapter, I introduce a solution approach for solving the problem described in the third chapter. This approach uses previous scientific results, and I propose extensions based on those contributions. The solution procedure consists of two main components: (1) an extended, flexibly controllable constructive scheduling procedure and (2) a metaheuristic search, which can drive and apply the constructive scheduling procedure as an embedded method (constructive simulator). I also present an extended evolutionary algorithm, which provides more efficient solutions compared to the original method for certain production scheduling problems.

The fifth chapter summarizes the results of my software architecture design and application development work carried out to implement the solution approach. The chapter aims to provide a conceptual overview of the implementation, covering the compositional structure of the developed software and a proposed object-oriented implementation.

In the sixth chapter, I demonstrate the applicability of the model on various known production scheduling and benchmark problems. I describe the applied measurement methods, the numerical results, and the conclusions drawn from these results. For testing the multi-project extension, I created a test dataset based on known benchmark problems and examined

the flexibility and efficiency of the solution method on these problems, conducting tests according to different objective function configurations.

The seventh chapter of the dissertation summarizes the new scientific results in the form of theses. In the following chapters, I present my own scientific publications, the list of references, and the lists of the figures and tables included in the dissertation. The appendix contains additional details related to the highlighted numerical results presented in the dissertation.

Köszönetnyilvánítás

PhD értekezésem elkészítése hosszú folyamat és munka eredménye, amely során számos személy támogatott és bíztatott, hogy elérjem a kitűzött céljaimat. Értekezésemben szeretném kifejezni őszinte hálámat mindazoknak, akik különböző módokon hozzájárultak a munkám eredményéhez.

Elsőként szeretném köszönetemet kifejezni szeretett feleségemnek, Reninek, aki a doktori iskolába jelentkezésem óta kitartóan mellettem állt és rendíthetetlenül hitt bennem. Nemcsak a folyamatos támogatása, hanem a türelme, megértése és erőfeszítése is elengedhetetlen volt ahhoz, hogy átvészeljem a nehezebb időszakokat. Megadta azt a környezetet, hogy lehetőségem legyen a kutatási témával foglalkozni, miközben rengeteg feladatot és munkát vállalt magára.

Külön mérhetetlen hálával tartozom konzulensemnek, dr. Kulcsár Gyulának, akinek szakmai irányítása, baráti kedvessége és személyes támogatása meghatározó volt a kutatásom sikeres véghezvitelében. Mind az inspiráló ötletei és elhivatottsága a tématerület iránt, mind a türelme és rendkívül értékes kritikái nagyban hozzájárultak ahhoz, hogy az értekezésem elérje a végső formáját. Minden alkalommal számíthattam az útmutatásaira és folyamatos korrekcióira, amelyeket nagyra értékelek. Megalkuvást nem tűrő alapossága és precizitása meghatározó minta lett számomra.

Szeretném kifejezni köszönetemet korábbi konzulensemnek, dr. Hornyák Olivérnek is, aki a kutatásom kezdeti szakaszában támogatott. Az ő munkája és javaslatai segítettek abban, hogy a kutatási irányom letisztuljon, megismerjem a kutatói feladatok fő szempontjait és kihívásait.

Továbbá köszönöm dr. Nehéz Károly tanszékvezető támogatását, aki mindvégig megteremtette azokat a szakmai feltételeket, amelyek lehetővé tették, hogy a kutatásom zavartalanul folyhasson és az oktatási feladataimon keresztül a tudományos ismeretanyag hatékony átadásának mikéntjét elsajátítsam.

Végül, de nem utolsósorban szeretném köszönetet mondani barátaimnak, kollégáimnak és további szakmai vezetőimnek, akik szintén mindig készségesen segítettek és kedvességükkel bátorítottak az úton.

Ezt a munkát szeretett gyermekeimnek, Botondnak, Marcellnak és Majának ajánlom, akik életvidámságukkal és szeretetükkel mindig új lendületet adtak nekem. Ők hárman minden nap emlékeztettek arra, hogy miért érdemes kitartóan dolgozni és törekedni egy jobb holnapra.

1 Bevezetés

Az idővel való megfelelő gazdálkodás rendkívül fontos szerepet játszik a különböző rendszerekben végbemenő folyamatok tervezésében és irányításában. A folyamatok nagyon sokfélék és változatosak lehetnek a végrehajtási környezettől függően. Tervezési és irányítási szempontból egy jellegzetes változatot jelent a projekt-orientált szemlélet. Egy projekt alapvető jellemzője, hogy egymással kapcsolatban álló tevékenységekből áll és a tevékenységek elvégzéséhez meghatározott erőforrásokra van szükség. A gyakorlatban a rendelkezésre álló erőforrások rendszerint korlátozottak, ezért a tevékenységek hatékony végrehajtása jól megtervezett ütemterveket igényel. Ezek létrehozására ütemezési modelleket és ütemezési algoritmusokat célszerű használni. A projektütemezési funkció hatékony megvalósításához olyan szoftverre van szükség, amely a felmerülő ütemezési feladattípusokat képes értelmezni és hatékonyan megoldani. Az ütemezéssel kapcsolatos tudományos kutatómunkák erőteljesen hozzájárulnak a megfelelő alkalmazások egyre hatékonyabb megoldó motorjainak fejlesztéséhez.

Tudományos kutatómunka esetén meg kell ismernünk a tématerület jellemzőit, kutatásának történetét és további lehetőségeit, akár alapkutatásról, elméleti kutatásról vagy ipari gyakorlatban felmerülő kihívásokra választ adó tudományos tevékenységről van szó. Ezt követően -, amennyiben azt tapasztaljuk, hogy a meglévő tudás, vagy eszközkészlet nem elegendő az azonosított kihívások megfelelő megválaszolására - kitűzhetjük a kutatásunk célját. A célok jó megfogalmazása segíti a kutatási irányok rendezettségét és a hatékonyságot. A célok kijelölése után meg lehet fogalmazni azokat a tudományos kutatási módszereket és eljárásokat, amelyeket használva a kitűzött célt el szeretnénk érni. Végül tudományos módszerekkel ellenőriznünk kell, hogy az általunk végzett munka elérte-e a kitűzött célokat, az elvégzett munka által milyen eredményekre és következtetésre jutottunk.

1.1 A kutatás tématerülete

Napjainkban megfigyelhető, hogy a projekt alapú tervezés és végrehajtás egyre nagyobb szerepet kap a gyártással kapcsolatos folyamatok minden területén. A termék életciklus menedzsment rendszerek (*Product Lifecycle Management – PLM*) átfogó megoldást kínálnak egy termék teljes életciklusának kezelésére kezdve a koncepcionális ötleteléstől, a termék- és termelés-tervezésen át a gyártásba bocsájtásáig, melyet a terméktámogatás és végül a termék kivezetése követ. Ezek a PLM rendszerek rendszerint tartalmaznak saját, vagy integrált projekt kezelő (*Project Management – PM*) megoldásokat, melyek segítségével az üzleti folyamatok szervezésére új eszközkészlet áll rendelkezésre. A projekt fogalom eredetileg nem a terméktervezés, vagy a termékgyártás sajátja, azonban a megfogalmazott jellemzői alapján a gyakorlatban jól alkalmazhatónak bizonyult, ahogy az más alkalmazási területeken is megfigyelhető [1]. A projekt tekinthető egy időben jól körbe határolt, jól meghatározott cél elérésének érdekében végzett tevékenységek és cselekvések összességének [2]. Projekt alapú feladatszervezést alkalmazó cégek egyidejűleg több projektet is végrehajtanak. A projektek gyakran osztoznak közös erőforrásokon és ez az erőforrásmegosztás vezethet olyan

versenyhelyzethez, amikor a párhuzamosan futó projektek által támasztott igények időben egyszerre már nem teljesíthetőek. A versenyhelyzet feloldása egy olyan feladat-végrehajtási sorrend kialakításával történik, ami lehetővé teszi az egy időben támasztott igények korlátokat figyelembe vevő teljesítését. Minden projekt saját erőforráskészlettel, célokkal és korlátokkal rendelkezik. Ez a projektszintű meghatározottság egy cég szintű optimális ütemterv kialakítását nehézzé teszi, különösen abban az esetben, ha a projektek egymástól nem független módon hajthatóak végre a megosztott erőforrások, vagy feladatok végrehajtásának sorrendi megkötése miatt.

A globalizálódó, egyre nagyobb versenyhelyzettel szembesülő vállalatok számára kiemelkedően fontos, hogy a termékmenedzsmenttel és termék gyártással kapcsolatos folyamatok hatékonysága és rugalmassága minél magasabb szintet érjen el. A szorosan összefüggő ellátási láncok, az alacsonyan tartott készletszintek, a nagy fokú termék-variancia és rugalmas gyártás miatt a végrehajtásnak alkalmasnak kell lennie arra, hogy egy megváltozott környezeti tulajdonságra hatékonyan és rugalmasan tudjon reagálni. Ez kiterjedhet az erőforrások elérhetőségének megváltozására, a kezdetben kitűzött célok közötti fontossági arányok módosítására, vagy akár a kitűzött célok teljes módosítására is. Előfordulhat, hogy olyan új üzleti cél megvalósítását, vagy korlátozó feltétel figyelembevételét kell támogatni, amelyre az eredeti rendszer nem volt felkészítve. Ilyen esetben fontos, hogy az alkalmazott projektütemező rendszer kiterjeszthető és módosítható legyen.

1.2 Kitűzött kutatási célok

A többprojektes menedzselési környezetekben jelentkező igények alapján szükségesnek tartom olyan ütemezési modell kidolgozását, amely lehetővé teszi valós üzleti problémák hatékony kezelését azáltal, hogy figyelembe veszi sok projekt egyedileg meghatározott céljait és igényeit a részletes végrehajtási ütemtervek generálásakor. A modell alapján olyan ütemező módszer kifejlesztését is szükségesnek tartom, amely hatékonyan támogatja sok, különböző prioritású célfüggvény együttes optimalizálását a rendelkezésre álló erőforrások és a végrehajtandó projektek egyedi korlátfeltételeinek egyidejű betartása mellett. A kutatás kitűzött céljai:

1. Az erőforráskorlátos, többcélú, többprojektes ütemezési feladatok ismert modelljeinek és ütemező eljárásainak vizsgálata.
2. Többprojektes környezetben jelentkező ütemezési feladatok matematikai modelljeinek kutatása és fejlesztése. Kiemelt szempont, hogy a modell kezeljen rugalmasan definiálható és eltérő optimalizálási iránnyal rendelkező célfüggvényeket.
3. Ütemezési eljárások kutatása, ütemezési módszertan és algoritmusok fejlesztése.

1.3 Alkalmazott tudományos kutatási módszerek

Kutatómunkám a szakirodalomban publikált modellek és eljárások megismerésén alapul. A kutatómunka része, hogy a tématerület bemutatásánál vázolt kihívások az ismert modellekkel hogyan kezelhetőek, mely esetben teljesítik az elvárt célokat és hol mutatnak hiányosságot. Ezek alapján témavezetőmmel felmértem, hogy milyen új kutatási irányokra

adhat lehetőséget a projektütemezés témaköre. Ezeket felhasználva kiválasztottam azt az alapmodellt, amelyet kiindulási modellként használhatok. Mindezek után lefektettem egy teljes matematikai modellt, amely tartalmazza a továbbfejlesztési irányként megfogalmazott többcélúságot és a célfüggvények dinamikus kiterjeszthetőségét.

A modell felállítása után megvizsgáltam a korábbi modellek megoldására tett javaslatokat. Igazolt, hogy a kiindulásként választott alapprobléma az NP nehéz feladatok körébe tartozik, azaz megoldásukra nem lehetséges polinomiális számítási idejű algoritmust készíteni. Az a kutatás elején világos lett, hogy kiterjesztett modell esetén a korábbi modellek speciális esetként modellezhetőek, így a Karp-redukciót alkalmazva az új modell is NP nehéz osztályba tartozik. Ezt a dolgozatban a futási eredmények fejezetben mutatom be. A feladat megoldásához egyrészt kereső eljárásokat kutattam, illetve a felépítő elvű eljárások alkalmazhatóságát is megvizsgáltam. Végül a két módszertan előnyeit együttesen kihasználó ütemezési megközelítés lehetőségét vizsgáltam meg.

A kidolgozott modell és eljárások alkalmazhatóságát szoftveres adaptáción keresztül ellenőriztem. Készítettem egy szoftvert, amelynek adatmodellje a kidolgozott matematikai modellre alapul. A szoftver tartalmaz egy heurisztikus keresőből és egy felépítő eljárásból álló modult, melyek alkalmazzák a többcélú optimalizálás során kidolgozott számítási eljárást.

A szoftver funkcionális tesztelése után teljesítménytesztet futtattam generált teszt adatsorokon és ismert benchmark feladatokon. A tézisek érvényességét az itt mért mutatók alapján igazoltam.

A tudományos kutatómunka eredményeit magyar és idegen nyelvű tudományos konferenciákon mutattam be, illetve tudományos folyóiratokban publikáltam.

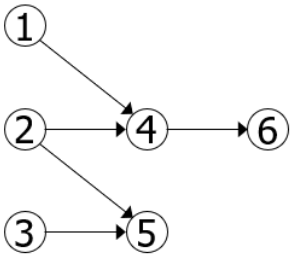
2 Irodalmi áttekintés

A szakirodalomban nagy számú könyv, folyóiratcikk és kutatási eredményt közlő konferencia közlemény található, melyek az ütemezési problémák témakörével foglalkoznak. Számos kutató javasolt új modelleket és algoritmusokat a különböző ipari optimalizálási problémák megoldására. Ebbe a kategóriába tartozik például a járműútvonal-optimalizálás [3], a munkaterhelés-szabályozás [4], a folyamatirányítás [5], a feladatok munkásokhoz rendelése [6], a projekt előrehaladásának értékelése [7], a munkaterület-elrendezés optimalizálása [8], az ellátási lánc optimalizálása [9], a gyártástervezés [10], a termeléstervezés és -irányítás [11], a termelésütemezés [12] és sok más probléma. A bevezetőben ismertetett kutatási témám szorosan kapcsolódik ezekhez a területekhez. Egy új, kiterjesztett optimalizálási modell kidolgozását határoztam meg. A kiterjesztett modell alapjául az erőforráskorlátos projektütemezési problémák modelljét (*Resource-constrained Project Scheduling Problem – RCPSP*) tekintettem.

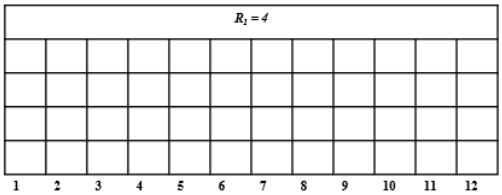
Az eredeti RCPSP probléma egy olyan projektütemezési feladat, amelynek célja egy adott projekt tevékenységeinek időbeli megtervezése, figyelembe véve a feladatok között meghatározott előfeltétel-korlátokat és az előre meghatározott erőforrás-korlátokat. A projekt végrehajtandó feladatokból áll. A projekt végrehajtásához erőforrások, előre meghatározott maximális kapacitással állnak rendelkezésre. Minden feladat esetén ismert a végrehajtási ideje és a végrehajtásához szükséges erőforrás szükséglet minden rendelkezésre álló erőforrás tekintetében. Az optimalizálási feladat lényege, hogy meghatározzuk, mikor kezdődjenek az egyes feladatok úgy, hogy a projekt teljes befejezési ideje a lehető legkisebb legyen és az ütemezés ne sértse az erőforrások rendelkezésre álló kapacitáskorlátját. Egy szemléltető példát mutat be az 1. ábra.

Feladat	Végrehajtási idő	Előfeltétel feladat	Erőforrásigény	
			R_1	R_2
1	2	-	3	2
2	3	-	1	1
3	1	-	2	1
4	4	1,2	2	1
5	2	2,3	3	2
6	1	4	3	1

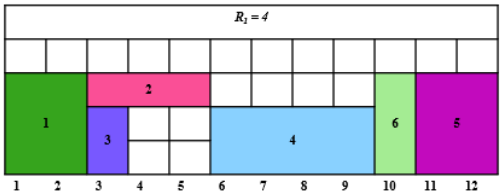
a. Végrehajtandó feladatok előfeltételei és erőforrásigényei



c. Feladatok precedencia gráfja



b. Rendelkezésre álló erőforrások és kapacitáskorlátjuk



d. Egy megvalósítható ütemterv

1. ábra - Szemléltető RCPSP feladat

Az RCPSP probléma első matematikai leírását Pritsker és társai fogalmazták meg 1969-ben [13]. A modell elsőként vezette be és alkalmazta a projekt teljes időtartama (*makespan*) fogalmat, amely célfüggvényként fejezi ki egy projekt utolsó feladatának a befejezési idejét. Blazewicz és társai 1983-ban matematikai redukciós megközelítéssel bebizonyították, hogy az RCPSP problémák az NP nehéz problémák osztályába tartoznak [14].

Az RCPSP projektütemezési feladatok megoldására több matematikai elveket alkalmazó optimalizációs eszköz és keretrendszer létezik. Az IBM ILOG CPLEX Optimization Studio egy korszerű eszköz, amely leginkább a lineáris programozási (*Linear Programming* – LT) és a vegyes egészértékű lineáris programozási (*Mixed-Integer Linear Programming* - MILP) problémák megoldására specializálódott. Megoldó algoritmusként a szimplex-módszert, valamint a branch-and-bound és a branch-and-cut algoritmusokat alkalmazza. A Gurobi Optimizer egy másik elterjed, matematikai elvekre épülő optimalizációs szoftver, amely párhuzamosított számítást és heurisztikákat is alkalmaz, amelyek nagyobb méretű problémák hatékony kezelését is lehetővé teszik. A korlátozásprogramozás (*Constraint Programming* - CP) alapú eszközök, például a Google OR-Tools vagy a MiniZinc, kombinatorikus és erőforrás-kezelési problémák esetén erősek. Ezek az eszközök diszkrét változókkal és logikai korlátozásokkal dolgoznak, gyakran propagációs és keresési technikákat alkalmazva. Az OptaPlanner, egy nyílt forráskódú Java keretrendszer, amely metaheurisztikákra alapozva kezeli hatékonyan a projektütemezési feladatokat.

Az RCPSP problémák méretének megoldhatósága nagymértékben függ a választott módszertől és a probléma specifikus jellemzőitől, úgy mint az ütemezendő feladatok száma, rendelkezésre álló erőforrások típusa és a definiált korlátok, valamint az aktuálisan elérhető számítási teljesítménytől. A tapasztalat azt mutatja, hogy az MILP és CP alapú módszerek (CPLEX, Gurobi) kiemelkedően pontosak, azonban a praktikusan megoldható feladatok mérete 40-50 feladatra és 3-10 erőforrásra korlátozódik. A számítási idő a tevékenységek és az erőforrások számával exponenciálisan nő, ezért a fő alkalmazási területek a kis- és közepes méretű projektekre összpontosulnak. A korlátozásprogramozásra és a visszalépéses keresésre (*backtracking*) alapuló algoritmusok hatékonyak a feladat-függőségek, illetve a komplex erőforrás-korlátos feladatok kezelésére. Az elérhető számítási kapacitás tekintetében nagy méretű problémák esetén ma a heurisztikus módszereket alkalmazó megközelítések tekinthetőek az egyetlen reális opciónak.

Az RCPSP probléma megfogalmazása óta több összefoglaló cikk került publikálásra, melyekben a feladat különböző megoldási lehetőségeit vizsgálják [15], [16], [17], [18]. Már az eredeti RCPSP modell is nagy kutatási múltra tekint vissza és eredeti formájában is nagyon sok ütemezési feladat leképezésére alkalmas, ennek ellenére a gyakorlati alkalmazások további kiterjesztéseket tettek szükségessé. Az ismert RCPSP kiterjesztésekről Hartman és Briskorn ad egy mély és összefoglaló jellegű áttekintést [19]. Hartman és Briskorn az eredeti modell variánsait és kiterjesztéseit az alábbi szempontok szerint osztályozták: (1) a projektet alkotó feladatok általánosítása; (2) alternáló megelőzési feltételek és feladat kapcsolati karakterisztikák bevezetése; (3) többprojektes szempontok hozzáadása. A hivatkozott cikk jól

strukturált módon foglalja össze a modellek kiterjesztésének módjait, valamint a kiterjesztésekre adott megoldási megközelítéseket.

Ezek a kiterjesztési irányok alapvetően a tématerület nyitott kérdései köré vannak csoportosítva. Figyelembe véve a legújabb kutatási irányokat és a gyakorlati alkalmazásban felmerülő igényeket, kutatási területem Hartman és Briskorn szerinti kategorizálás mindhárom csoportjába sorolható, mivel a feladatok dinamikus attribútumait, több projekthez való hozzárendelhetőségét, többcélúságot és többprojektes kiterjesztést is vizsgál.

Többcélúság egyidejű kezelését más kutatások is felvetették és vizsgálták már [20]. Általánosságban elmondható, hogy a kutatók az egynél több célfüggvényt egyszerre alkalmazó problémákat a tudományos irodalomban többcélú optimalizálási problémaként (*multi-objective optimization problem*) nevezik [21]. Az alkalmazott célfüggvények száma szignifikáns emelkedést mutat. A gyakorlati feladatok megoldására a kutatók újabb és újabb célfüggvényeket dolgoztak ki. A célfüggvények számának növekedése az ütemezési problémák megoldásának tekintetében egyben új kihívást is jelent, mely megoldására új megoldási módszerek kidolgozására van szükség. Ezek az új kihívások egyrészt magukban foglalják ahogyan új megoldási jelölteket generálunk, illetve azt, hogy a különböző megoldásokat praktikusan alkalmazható teljesítménymutatók alapján miként tudjuk egymáshoz hasonlítani [22]. A többcélú optimalizálási problémák megoldására a kutatók különböző módszereket alkalmaznak, melyek alapulhatnak a már meglévő megoldási módszereken, jelenthetnek kiterjesztett módszert, új módszert vagy a korábbi módszerek hibrid alkalmazását.

A többcélú projekt ütemezési probléma megoldására a kutatók széles körűen alkalmazzák az alábbi három minta valamelyikét: (1) az egyes célfüggvények súlyozott lineáris kombinációját; (2) Pareto-elvet; (3) sorrendezett, vagy priorizált célfüggvényeket. A projekt teljes időtartamának és a költségek kombinációjának minimalizálását fogalmazta meg és alkalmazta Dai és társai [23], valamint Schnabel és társai [24]. Az egyedi projektszintű, valamint az egyesített projekt portfóliószintű célok optimalizálására mutattak módszert Tirkolaee és társai [25]. A Pareto-hatékonyság elve néhány célfüggvény esetén azt vizsgálja, hogy mely célfüggvények módosítják leginkább az ütemezés teljesítménymutatóit (*Key Performance Indicators – KPI*). Tabrizi és társai két célfüggvényt fogalmaztak meg, melyben az első a projekt teljesítési idejének és a projekt határidőjének kombinációja, a második pedig az úgynevezett rendelések ökológiai hatása (*ecological impact of order*) volt [26]. A megfogalmazott célfüggvényeket anyagbeszerzési feladatok esetén alkalmazták. Dridi és társai a projekt teljes időtartamát és a költségek minimalizálását vizsgálta a megújuló erőforrások tekintetében [27].

A többcélú optimalizálási eljárások több módon is kategorizálhatóak. Alkalmazhatunk változatosság alapú (*diversity-based*), indikátor alapú (*indicator-based*), nyugodt-domináló elvű (*relaxed-dominance-based*), előnyben részesítés alapú (*preference-based*), összesítés alapú (*aggregation-based*), referenciahalmaz alapú (*reference-set-based*) és kiterjesztő-szűkítő elvű (*dimensionality-reduction-based*) csoportosítást. Mane és Rao [28], Taha [20], valamint Manea és Narsingraoa [29] adtak munkájukban összefoglaló áttekintést a többcélú

optimalizálási problémákról és algoritmusokról. Manea és Narsingraoa [29] nem csak áttekintették és rendszerezték az ismert többcélú optimalizálási eljárásokat, hanem egy új, a Jaya algoritmust kiegészítő, kautikus-jellegű (*chaotic-based*) megközelítést is bemutattak a többcélú optimalizálási problémák megoldására.

Az általam vizsgált projektütemezési problémák többcélú optimalizálási vetületeinek kezeléséhez a Kulcsár és Erdély által javasolt relatív változásra alapozott szemléletet vettem alapul [30]. A javasolt megközelítés eredményesen beépült doktori értekezésekbe [31], [32], valamint több gyakorlati alkalmazási példa is alátámasztotta a relatív változásra alapozott megközelítés előnyeit [33], [34], [35], [36], [37], [38].

A többcélú optimalizálás mellett egy másik általam követett fontos kutatási irány volt a projektütemezési modellek fejlődése. Az alábbi cikkek összefoglaló áttekintést adnak a projekt ütemezési problémák fő variánsairól [39], [40], és [41]. Schwindt and Zimmermann kétkötetes könyvben foglalja össze a projektütemezés fontos modelljeit és megoldási módszereit [42], [43]. Az idő előrehaladásával a témával foglalkozó kutatók az RCPSP probléma további új kiterjesztéseit publikálták. Erről ad jó áttekintést például a [44], és a [19] munka.

Nagyon sok klasszikus gyártásütemezési problémára tekinthetünk az RCPSP, vagy az RCPSP többprojektes kiterjesztésének (*Resource Constrained Multi-Project Scheduling Problem – RCMPSP*) speciális eseteként. Például az egy gépes (1), párhuzamos gépes (P), flow shop (F), flexible flow shop (FF), job shop (J) és flexible job shop (FJ) ütemezési problémák leképezhetőek az RCPSP vagy az RCMPSP modellre. A leképzés során a gyártási művelet (*operation*) a project egy aktivitására képezhető le, egy erőforrás típus áll rendelkezésre, amelyben a feladattípustól függően vagy egy, vagy több ugyanabba a munkagépcsoportba sorolt munkagép található és a munkagépek szükségesek a gyártási műveletek végrehajtására. Ezzel a megközelítéssel egy gyártási feladat egy projekttel is reprezentálható.

A job shop (J) és flexible job shop (FJ) problémák megoldására több metaheurisztikán alapuló kereső algoritmust alkalmaztak, például genetikus algoritmusokat ([45], [46], [47], [48], [49]), szimulált hűlést ([50], [51]) és tabu keresést ([52], [53], [54]). Ezek mellett a természet inspirálta evolúciós metaheurisztikákat is gyakran alkalmaznak az optimalizálási probléma megoldására, úgy mint a részecske-raj algoritmus (*Particle swarm – PSO*) [55], a hangya kolónia algoritmus (*Ant Colony – ACO*) [56], a méh kolónia algoritmus (*Bee Colony Algorithm – BCA*) [57], a bakteriális algoritmus (*Bacteria Algoritm – BA*) [58] és a denevér algoritmus (*Bat Algorithm – BAT*) [59]. Ezenkívül ebbe a csoportba sorolható még a szentjános bogár algoritmus (*Firefly Algorithm – FFA*) [60], a fekete lyuk algoritmus (*Black Hole Algorithm – BHO*) [61], virág beporzás algoritmus (*Flower Pollination Algorithm – FPA*) [62], és a harmónia keresés (*Harmony Search – HS*) [63].

Az irodalomban különböző heurisztikákat is javasoltak az RCPSP megoldására, melyeket három fő csoportba sorolhatunk. Az első csoportba az egyutas heurisztikák tartoznak, a második csoportba a több utas heurisztikák, míg a harmadik csoportot a metaheurisztikák alkotják. Az egy- és többutas heurisztikák előnye, hogy gyorsan tudnak végrehajtható

ütemezést szolgáltatni. Ezek a módszerek ütemezést generáló sémákat (*schedule generation scheme – GS*) használnak. Az ütemezési sémák soros és párhuzamos verziója alkalmazható olyan módon, hogy az ütemezés felépítése során egy vagy több prioritással rendelkező szabályt alkalmazunk. A harmadik csoportot alkotó metaheurisztikák különböző variánsai összetett algoritmusokat alkalmaznak, hogy a lehetséges megoldások terét (*searching space*) hatékonyan felkutassák. A metaheurisztika alapú keresők fő előnye, hogy a szolgáltatott ütemezés jó minőségű, amely alatt azt értjük, hogy az elérhető optimumhoz jól közelít. A metaheurisztikus keresők hátránya, hogy nagy számítási igényt támasztanak és az ütemtervet lassabban tudják előállítani.

Pellerin és társai hívták fel a figyelmet arra a trendre, hogy az elmúlt két évtizedben a metaheurisztikus optimalizálás egyre inkább elmozdult a tisztán egyetlen metaheurisztikát alkalmazó módszerek alkalmazásától a kevert vagy hibrid módszerek felé. Hibrid módszer esetén különböző metaheurisztikák együttes alkalmazása hatékonyabban állíthat elő végrehajtható ütemezéseket, melyek a lehetséges optimumot jobban közelítik. A vizsgált hibrid megközelítési módszerek a [18] munkában kerültek kifejtésre és összehasonlításra. A különböző hibrid módszerek összehasonlításának alapja az RCPSP problémák kutatók körében jól ismert PSLIB benchmark adatain alapulnak.

A tématerület aktív kutatásai eredményeként Vanhoucke és Coelho tanulmányaikban a nagyméretű RCPSP ütemezési problémák kezelését vizsgálták [64], [65]. Xue és társai egy branch-and-bound módszert hasonlítottak össze CPLEX és genetikus algoritmussal erőforráskorlátos proaktív projektütemezési feladatokon [66].

Az RCPSP és RCMPSP a projektmenedzsment alapvető problémáinak tekinthetők. A modellek és megoldó módszereik fontos szerepet játszanak mind a kutatók, mind a gyakorlati alkalmazások számára. A legkülönbözőbb alkalmazási területeken megtaláljuk a projektmenedzsment támogatási rendszerektől a gyártástervezésen át a gyártásütemezésig.

3 A kiterjesztett projektütemezési probléma modellezése

Gyakorlati feladatok megoldása közben gyakori módszer, hogy egy konkrét tématerület – vagy adott esetben akár nagyon eltérő területek – fogalmait egy általánosított terminológiára és modellre képezünk le. Ennek előnye, hogy az általánosított modell által leírt problémát megoldó eljárások és módszerek később több területen is alkalmazhatóvá válnak, mivel az általánosított modell megoldása szempontjából az alkalmazási terület eredeti fogalmai már nem bírnak további relevanciával. A megközelítés hátránya, hogy az alkalmazott leképzést a valós probléma fogalmai és az általános modell terminológiája között minden esetben el kell végezni; a valós problémát előbb leképezzük a modellre, majd a modellre kidolgozott megoldó eljárások eredményeit vissza kell alakítani, hogy azok így a konkrét felhasználási területen is alkalmazhatóvá váljanak.

Projektütemezési területen a szakirodalomban, valamint a menedzsment tudományok különböző területein a leggyakrabban használt fogalmak a következők: tevékenység (*activity*), projekt (*project*) és erőforrás (*resource*). Műszaki rendszerek esetén a tevékenység és aktivitás helyett előnyben részesítjük a művelet és operáció fogalmakat, például amikor egy gyártási terv egy adott technológiai eljárása kerül meghatározásra. Ennek oka, hogy a technológiai folyamatokban a művelet általában az erőforrás-igény szempontjából elemi egységnek tekinthető és egyaránt vonatkozhat emberi vagy gépi munkavégzésre. Az aktivitás kifejezés inkább a menedzsment tudományok modelljeiben jelenik meg általános tevékenység megjelöléseként. A diszkrét termelési rendszerek irányítási feladataiban elterjedten alkalmazzák a munka (job) fogalmát is, amely több művelet együttesének megnevezésére szolgál. Például: adott számú egyforma munkadarab legyártása meghatározott műveletek adott sorrendben történő végrehajtásával. A menedzsment szemléletben a projekt kifejezést használják a valamilyen szempontból összetartozó tevékenységek együttesének megnevezésére.

A kiterjesztett ütemezési modellt mind a műszaki rendszerekben, mind az általánosabb ütemezési feladatokkal terhelt területeken alkalmazni szeretném. A terminológiai egyértelműség érdekében az elkészített modellemben a *feladat (task)* kifejezést használom egységesen az aktivitás, tevékenység, művelet és operáció helyett. Fontos azonban hangsúlyozni, hogy ez a döntés csupán az elnevezéseket érinti, mivel az új modell az egyéb fogalmakat használó szakterületeken is egyenértékűen alkalmazható marad.

A feladat (task) egy tetszőleges, de egyértelműen definiált elemi vagy összetett, végrehajtandó folyamatot jelent. A feladat meghatározása és részletezettsége attól függ, hogy milyen területen alkalmazzuk az ütemezési modellt. Például a gyártásütemezés esetén a feladat lehet egy anyagmegmunkálási lépés egy adott gépen, egy gyártási sorozat legyártása egy termelői soron, vagy egy kézi megmunkálási lépés végrehajtása egy szakképzett dolgozó által. A sportesemény-szervezés esetén feladatnak tekinthetjük a nézők beengedését az esemény helyszínére, a versengő csapatok pályára lépésének engedélyezését, vagy a helyszín takarítását. A feladatnak lehet erőforrásigénye, amely definiálja, hogy a végrehajtásához különböző erőforrásokat kell felhasználni egyidejűleg. Az egyes erőforrásokat erőforrástípusokba (*resource type*) szervezhetjük. Egy erőforrástípus fogja össze azokat a konkrét erőforrásokat,

amelyek egy adott szempontból a feladatok végrehajtására alkalmazhatók. A feladatot és az erőforrástípust a modellben egy-egy entitásnak tekintem.

Az alkalmazási terület fogalmainak további jellemzőit a modellben támogatott entitások attribútumaira képezhetjük le. Minél több jól megfogalmazott attribútum áll rendelkezésre, annál részletezettebben képezhető le a valós üzleti feladat. A modellezett üzleti probléma elemei közötti összefüggéseket a modellben definiált elemek kapcsolatára képezzük le.

A legtöbb gyakorlati feladatnál felmerülnek különböző korlátok, amelyeket figyelembe kell venni a megvalósítás során. Ennek megfelelően a modellnek támogatnia kell a feladat releváns korlátjainak hatékony leképzését. Korlátnak tekinthető például egy gyártási tároló maximális kapacitása, egy adott feladat egy adott gépen történő végrehajtásához szükséges idő, a különböző gépek beállítási ideje, egy gép karbantartási feladatának elvégzéséhez szükséges szakképzett munkások száma, egy szoftverfejlesztési feladat végrehajtásához szükséges dolgozók képesítése és létszáma, egy sportesemény esetén a biztonsági ellenőrzések elvégzési idejének szükségessége és így tovább. Ezeket a megkötéseket a modellben előre meghatározott korlátozási típusokra képeztem le, melyeket a megoldó eljárásnak figyelembe kell vennie úgy, hogy a problémára adott megoldás ne sértsen semmilyen korlátozást.

Azt, hogy egy modellezett problémára adott megoldás mennyire jó, az adott üzleti terület sajátosságai határozzák meg. Fontos előre meghatározni, hogy egy adott tervezési folyamatban melyek azok a fő szempontok és célok, amelyet figyelembe szeretnénk venni. Például a gyártásütemezési feladatok esetén általában olyan célokat tűzhetünk ki, mint a magasabb gépkihasználtság, a kisebb átfutási idő vagy a minimális készletszint. Meghatározhatunk egyetlen célt, de akár több cél együttes teljesítésére is törekedhetünk. Gyakran előfordul, hogy a meghatározott célokat egyszerre nem lehet teljesíteni, ezért prioritásokat kell hozzájuk rendelni. A modellben a különböző lehetséges célokat és szempontokat célfüggvényekre képeztem le. A modell több célfüggvény együttes alkalmazását teszi lehetővé, amelyet rugalmasan lehet adaptálni az ütemezés során felmerülő célok és követelmények szerint. A matematikai modell a célfüggvények és korlátozások mellett az elsődleges döntési változókat is tartalmazza, amelyek az ütemezési feladat teljes megoldását reprezentálják.

Az irodalomban publikált RCPSP és RCMPSP modellek megvizsgálása után egy kiterjesztett modell létrehozására teszek javaslatot. Az új modellt erőforráskorlátos, többcélú, többprojektes ütemezési problémának neveztem el. A modellre az értekezésben az angol *Resource Constrained Multi-Objective Multi-Project Scheduling Problem* neve alapján az RCMOMPSP rövidítéssel hivatkozom.

3.1 Az RCMOMPSP modell

A fejezet további alfejezeteiben az RCMOMPSP modell formális, matematikai leírását adom meg. A matematikai leírás kitér az alapvető jelölésekre, képletekre és összefüggésekre, amelyek az alábbi sorrendben kerülnek ismertetésre: bemeneti adatok (3.2); a bemeneti adatokból származtatott másodlagos adatok (3.3); a bemeneti adatokra vonatkozó korlátozások

(3.4); az elsődleges döntési változók (3.5); az elsődleges döntési változókból származtatott minősítő értékek (3.6); a minősítő értékekre vonatkozó korlátok (3.7) és a célfüggvények (3.8). Erre az RCMOMPSP optimalizálási modellre vonatkozó 1. tézist az értekezés 7. fejezetében fogalmazom meg.

3.2 Bemeneti adatok

A feladat bemeneti adatait az alábbi jelölésrendszerrel határoztam meg.

P $\boldsymbol{P} = \{p_1, p_{2,} \dots, p_i, \dots, p_{NP}\}$, a végrehajtandó projektek (*project*) halmaza, p_i az i-edik sorszámú projekt és NP a végrehajtandó projektek ismert, véges száma.

T $\boldsymbol{T} = \{t_1, t_2, \dots, t_j, \dots, t_{NT}\}$, a végrehajtandó feladatok (*task*) halmaza. t_j a j -edik feladat és NT a végrehajtandó feladatok ismert, véges száma.

R $\boldsymbol{R} = \{r_1, r_2, \dots, r_k, \dots, r_{NR}\}$, a rendelkezésre álló erőforrástípusok (*resource type*) halmaza. r_k a k -adik erőforrástípus és NR a rendelkezésre álló erőforrástípusok ismert, véges száma.

$RC_k(\tau)$ az r_k erőforrástípus előre meghatározott elérhető kapacitása a τ időpontban (*resource type capacity at τ time*). Az $RC_k(\tau)$ egy előre definiált kapacitás-időfüggvénynek a τ időpontra vonatkozó aktuális értékét jelenti. Bemeneti adatokkal egyszerűen definiálható a függvény úgy, hogy időpont szerint növekvő sorrendben megadható az idő-érték párosok listája. Egy elempár a következő elempárig van érvényben és az utolsó elempár érvényes a végtelenig.

Az erőforrások megújulóak. Egy feladat végrehajtása nem fogyasztja az erőforrástípus kapacitását, hanem egy feladat végrehajtása csupán használja és foglalja a kapacitást. Egy feladat a végrehajtása kezdetekor lefoglalja a szükséges mennyiségű erőforrást, majd a végrehajtása után újra felszabadítja azt, hogy az egy másik feladat végrehajtására újra használható legyen. Egy erőforrástípus elérhető szabad kapacitása időben a már meglévő erőforrástípus-foglalások függvényében változik.

Virtuális erőforrástípusnak nevezem azt az erőforrás típust, amely nulla maximális kapacitással rendelkezik. A virtuális erőforrástípusokat párhuzamosan futó feladatok szinkronizálására vagy projekt mérföldkövek definiálására lehet célszerűen felhasználni.

Abban az esetben, ha egy erőforrástípus nem játszik az ütemezés szempontjából valós korlátozó szerepet (például feltehető, hogy mindig megfelelő mennyiségben rendelkezésre áll), akkor kapacitáskorlátot nem szükséges megadni. Abban az esetben, ha a modellben ezt az alkalmazási terület nem nulla kapacitásként mégis fel szeretné tüntetni, úgy egy alkalmasan választott nagy kapacitással ez megtehető, legegyszerűbben az összes ismert feladat összes ismert igényének összegeként. A modell szükség esetén kiterjeszthető úgy, hogy támogassa egy „nem korlátozó”, vagy „végtelen” érték megadását.

$rr_{j,k}$ $rr_{j,k} \epsilon \mathbb{Z}_0^+$, t_j feladat végrehajtásához szükséges kapacitás r_k erőforrástípusból (*required resource capacity*).

$pt_{j,k}$ $pt_{j,k} \epsilon \mathbb{Z}_0^+$, t_j feladat r_k erőforrástípuson szükséges végrehajtásához szükséges idő (*processing time*).

$\boldsymbol{TA}$ $\boldsymbol{TA} = \{(p_i, t_j) \mid \forall\, t_j \in \boldsymbol{T}\}$, feladatok és projektek összerendelésének halmaza (*task assignment*). A (p_i, t_j) összerendelés, azt fejezi ki, hogy a t_j task a p_i projekt része. Egy feladat egyidejűleg több projekt része is lehet. Ilyenkor minden érintett projekthez hozzá van rendelve az adott feladat.

$\boldsymbol{PRE}_j$ t_j feladat megelőző feladatainak halmaza (*predecessor tasks*). Megelőző feladatnak nevezzük azokat a feladatokat, amelyeket végre kell hajtani, mielőtt a t_j végrehajtását el lehet kezdeni.

$\boldsymbol{TPROP}$ $\boldsymbol{TPROP} = \{T_{rTime}, T_{dTime}, T_{prio}, \dots, T_{prop}\}$, a feladatoknak megadható egyedi lehetséges jellemzőinek előre meghatározott halmaza. Az egyedi jellemzők egy feladat különböző tulajdonságainak meghatározását teszik lehetővé. Jellemzőként modellezhető például a feladat legkorábbi indítási ideje (T_{rTime} – *task release time*), a feladat teljesítési határideje (T_{dTime} – *task due time*), a feladat fontossági mutatója (T_{prio} – *task priority*) és így tovább. A feladat jellemzőjére nincs típus megkötés, azt a feladat jellemzője határozza meg. A modellben a feladatok lehetséges jellemzői szabadon kiterjeszthetőek, azokra nincs modellezési korlát.

$\boldsymbol{TP}_j$ $\boldsymbol{TP}_j = \{T_{rTime,j}, T_{dTime,j}, T_{prio,j}, \dots, T_{prop,j}\}$, a t_j feladat előre meghatározott jellemzőinek halmaza.

$TP(t_j, T_{prop})$ jelöli azt a függvényt, amely egy t_j feladat T_{prop} jellemző által megadott értéket adja meg. A $TP(t_j, T_{prop})$ függvény „*nem létezik"* értékkel tér vissza, ha a T_{prop} tulajdonság a t_j feladat esetén nem meghatározott.

$\boldsymbol{PPROP}$ $\boldsymbol{PPROP} = \{P_{rTime}, P_{dTime}, P_{tCost}, P_{prio}, \dots, P_{prop}\}$, a projektek egyedi jellemzőinek lehetséges halmaza, amely egy projekt tulajdonságainak meghatározását teszi lehetővé. Projekt tulajdonsága lehet a projekt legkorábbi indítási ideje (P_{rTime} – *project release time*), a projekt határideje (P_{dTime} – *project due time*), a projekt késedelmességi költsége (P_{tCost} – *project tardiness cost*), a projekt fontossági mutatója (P_{prio} – *project priority*) és így tovább. A projekt jellemzőjére nincs típus megkötés, azt a projekt jellemzője határozza meg. A modellben a projektek jellemzői szabadon kiterjeszthetőek, azokra nincs modellezési korlát.

$\boldsymbol{PP}_i$ $\boldsymbol{PP}_i = \{P_{rTime,i}, P_{dTime,i}, P_{tCost,i}, P_{prio,i}, \dots, P_{prop,i}\}$, a p_i projekt meghatározott jellemzőinek halmaza.

$PP(p_i, P_{prop})$ jelöli azt a függvényt, amely egy p_i projekt P_{prop} jellemző által megadott értékét adja meg. A $PP(p_i, P_{prop})$ függvény „*nem létezik"* értékkel tér vissza, ha a P_{prop} tulajdonság a p_i projekt esetén nem meghatározott.

3.3 Bemeneti adatokból származtatott másodlagos bemeneti adatok

Az alábbi fejezetben további segédjelöléseket vezetek be a másodlagos, származtatott bemeneti adatokra, amely az RCMOMPSP probléma leírását és könnyebb kezelhetőségét teszik lehetővé. Minden másodlagos bemeneti adat a bemeneti adatokból számolható az alábbiak szerint:

$\boldsymbol{PT}_i$ $\boldsymbol{PT}_i = \{t_j | \ (p_i, t_j) \in \boldsymbol{TA}\}$, egy p_i projekthez rendelt feladatok halmaza.

$\boldsymbol{A}$ $\boldsymbol{A} = \{(t_j, t_{pre}) | \ t_j \in \boldsymbol{T}, t_{pre} \in \boldsymbol{PRE}_j\}$, $\boldsymbol{A}$ egy rendezett feladatpárokból (élekből) álló halmaz. Az $\boldsymbol{A}$ halmaz a megadott $\boldsymbol{PRE}_j$ és $\boldsymbol{T}$ halmazból származtatható.

G $G = (\boldsymbol{T}, \boldsymbol{A})$, A feladatok a feladatokhoz rendelt előfeltétel halmazokból egy irányított gráfba transzformálhatóak. A gráfban minden csúcs egy feladatot reprezentál és akkor vezet él a t_{pre} feladatot reprezentáló csúcsból a t_j feladatot reprezentáló csúcsba, ha a t_{pre} feladat a t_j előfeltétele: $(t_j, t_{pre}) \in \boldsymbol{A}$. Ezáltal a G gráf egy előfeltételi függőséget reprezentáló precedencia-gráf.

RC_k az r_k erőforrástípus maximális kapacitása: $RC_k = max_\tau(\ RC_k(\tau)\ )$.

3.4 A bemeneti adatokra vonatkozó korlátozások

Ebben az alfejezetben ismertetem azokat a bemeneti adatokra vonatkozó korlátozásokat, melyeket a probléma helyes értelmezése és kezelése érdekében teljesíteni kell. A korlátok formális leírása és bemutatása a következő:

- Nem létezik erőforrástípus negatív kapacitással.

$$RC_k(\tau) \geq 0 \mid \forall r_k \in \boldsymbol{R}, \forall \tau \quad (1)$$

- Az erőforrástípus függő végrehajtási (műveleti, feldolgozási) idő egyetlen feladat esetében sem lehet negatív.

$$pt_{j,k} \geq 0 \mid \forall t_j \in \boldsymbol{T} \quad (2)$$

- Bármely feladat erőforrástípus függő kapacitás igénye nem lehet negatív és amennyiben az erőforrástípus korlátozó jellegű, akkor nem lehet több, mint az erőforrás-típus maximális kapacitása.

$$0 \ \leq rr_{j,k} \leq \mathrm{RC}_k \mid \forall t_j \in \boldsymbol{T}, r_k \in \boldsymbol{R}, \mathrm{RC}_k > 0 \quad (3)$$

- Minden feladat legalább egy projekthez hozzá van rendelve. Ugyanaz a feladat tartozhat több projekthez is.

$$(p_i, t_j) \in \boldsymbol{TA} \mid \forall t_j \in \boldsymbol{T} \quad (4)$$

- Egy feladat nem lehet saját maga előfeltételi (megelőző) feladata.

$$\boldsymbol{PRE}_j \subseteq (\boldsymbol{T} \setminus \{t_j\}) \quad (5)$$

- A *G* gráfnak teljesítenie kell az irányított, körútmentes gráf (*Directed Acyclic Graph – DAG)* feltételeit.

3.5 Elsődleges döntési változók

Elsődleges döntési változónak az RCMOMPSP-ban leírt feladatok végrehajtásának kezdési időpontját tekintem. A feladatok kezdési időpontjai nem negatív egész számok. A feladatok kezdési időpontjai önmagukban meghatározzák az RCMOMPSP problémára adott megoldást, az ütemtervet. Az ilyen ütemtervben szereplő kezdési időpontokat az alkalmazási terület specifikus egységnyi időszakaira vetítjük le, hogy az ütemterv a gyakorlatban is alkalmazható legyen. Az egységnyi időszak lehet másodperc, perc, óra, nap, hét, egy agilis szoftverfejlesztési ciklus (*takt*), stb., amelyet a felhasználó tetszése szerint választhat, és így az időbeosztás a konkrét alkalmazás szempontjából rugalmasan adaptálható.

A meghatározott elsődleges döntési változók formai leírása:

S_j $S_j, (S_j \in \mathbb{Z}_0^+)$, A t_j feladat nem negatív egész számmal meghatározott kezdési időpontja.

SCH $SCH = (S_1, S_2, \dots, S_j, \dots, S_{NT})$, ütemezési vektor, amely az RCMOMPSP ütemezési probléma feladataihoz rendelt kezdési időpontokat tartalmazza.

3.6 Elsődleges döntési változókból származtatott minősítő értékek

A következőkben jelöléseket vezetek be az elsődleges döntési változókból és a bemeneti adatokból származtatott minősítő értékekre. Ezek segítik a korlátok és a célfüggvények könnyebb formalizálását.

C_j $C_j, (C_j \in \mathbb{Z}_0^+)$, A t_j feladat befejezési idejét reprezentáló nem negatív egész szám (*task completion time*). A t_j feladat befejezési idejét a feladat kezdési időpontjából és a végrehajtáshoz szükséges, erőforrástípus szerint ismert végrehajtási idők maximumából számolhatjuk az alábbi módon:

$$C_j = S_j + max_{k \in \boldsymbol{R}} (\{pt_{j,k}\}) . \quad (6)$$

C_{max} C_{max} jelöli a feladatok maximális befejezési idejét (*maximum completion time*). A maximális befejezési idő az RCMOMPSP probléma befejezési idejét is meghatározza. A maximális befejezési idő az alábbi módon számolható:

$$C_{max} = max_{j \in \boldsymbol{T}} (C_j) . \quad (7)$$

$C_{proj,i}$ $C_{proj,i}$ jelöli a p_i project befejezési idejét (*project completion time*). A projekt befejezési idejét a projekthez tartozó feladatok befejezési ideje alapján számíthatjuk:

$$C_{proj,i} = max_{j \in \boldsymbol{PT}_i} (\{C_j\}) . \quad (8)$$

$\boldsymbol{RT}(\tau)$ $\boldsymbol{RT}(\tau)$ (*running tasks at τ)* jelöli a τ időpontban végrehajtás alatt álló (éppen futó) feladatok halmazát. A τ időpont egy nem negatív egész szám és az időtől függő feladathalmaz az alábbi módon számolható:

$$\boldsymbol{RT}(\tau) = \{t_j \in \boldsymbol{T} \,|S_j \leq \tau \leq C_j, \tau \in \mathbb{Z}_0^+\} . \quad (9)$$

$\lambda_{j,k}(\tau)$ $\lambda_{j,k}(\tau)$ indikátor függvény, amely meghatározza, hogy a t_j feladat a τ időpontban a k erőforrás-típuson végrehajtás alatt van-e. Az indikátor függvény az alábbi módon számolható:

$$\lambda_{j,k}(\tau) = \begin{cases} 1, \ ha \ S_j \leq \tau \leq S_j + pt_{j,k} , \\ 0, \ egyébként. \end{cases} \quad (10)$$

$RL_k(\tau)$ $RL_k(\tau)$ függvény meghatározza az r_k erőforrástípus terhelését τ időpontban (*resource load at τ)*. A függvény definíciója:

$$RL_k(\tau) = \sum_{j \in \boldsymbol{RT}(\tau)} \lambda_{j,k}(\tau) * rr_{j,k} . \quad (11)$$

$\lambda_{pdelayed}(p_i)$ Indikátor függvény, amely meghatározza, hogy a p_i projekt késedelemmel rendelkezik-e. Az indikátor függvény az alábbi módon számolható:

$$\lambda_{pdelayed}(p_i) = \begin{cases} 1, \ ha \ C_{proj,i} > PP(p_i, P_{dTime}) \mid \exists \, PP(p_i, P_{dTime}) , \\ 0, \ egyébként. \end{cases} \quad (12)$$

$PL(p_i)$ $PL(p_i)$ jelöli a p_i projekt késését (*project lateness*). A projekt késése az alábbi módon számítható:

$$PL(p_i) = C_{proj,i} - PP(p_i, P_{dTime}) \mid \exists \, PP(p_i, P_{dTime}) . \quad (13)$$

$PT(p_i)$ $PT(p_i)$ jelöli a p_i projekt csúszását (*project tardiness*), amely az alábbi módon számítható:

$$PT(p_i) = \lambda_{pdelayed}(p_i) * PL(p_i) \mid \exists \, PP(p_i, P_{dTime}) . \quad (14)$$

$\lambda_{tdelayed}(t_j)$ Indikátor függvény, amely meghatározza, hogy a t_j feladat késedelemmel rendelkezik-e. Az indikátor függvény az alábbi módon számolható:

$$\lambda_{tdelayed}(t_j) = \begin{cases} 1, \ ha \ C_j > TP(t_j, T_{dTime}) \mid \exists \, TP(t_j, T_{dTime}) , \\ 0, \ egyébként. \end{cases} \quad (15)$$

$TL(t_j)$ $TL(t_j)$ jelöli a t_j feladat késését (*task lateness*). A feladat késése az alábbi módon számítható:

$$TL(t_j) = C_j - TP(t_j, T_{dTime}) \mid \exists\, TP(t_j, T_{dTime}) . \quad (16)$$

$TT(t_j)$ $TT(t_j)$ jelöli a t_j feladat csúszását (*task tardiness*), amely az alábbi módon számítható:

$$TT(t_j) = \lambda_{tdelayed}(t_j) * TL(t_j) \mid \exists\, TP(t_j, T_{dTime}) . \quad (17)$$

3.7 Megoldásra vonatkozó korlátozások

Ebben az alfejezetben a végrehajtható (*feasible*) megoldásokra vonatkozó korlátozásokat ismertetem. A korlátok formai leírásához a bemeneti adatokat, az elsődleges döntési változókat és az elsődleges döntési változókból származtatott minősítő értékeket alkalmazom. A korlátokat két kategóriába sorolom: (1) Feladat-orientált korlátozások; (2) Erőforrástípus-orientált korlátozások.

3.7.1 Feladat-orientált korlátozások

A feladatok kezdési (*start*) idejére az alábbi korlátok határozhatóak meg, amelyeket az ütemtervnek nem szabad megsértenie:

- Egy t_j feladat végrehajtása leghamarabb akkor kezdődhet meg, amikor t_j feladat összes megelőző (előfeltételi) feladata végre lett hajtva.

$$S_j \geq \max_{pre \in \boldsymbol{PRE}_j}(\{C_{pre}\}) \mid \forall t_j \in \boldsymbol{T} \quad (18)$$

- Ha t_j feladatnak van előre meghatározott legkorábbi indítási időpontja, akkor t_j feladat végrehajtása annál korábban nem kezdődhet meg.

$$S_j \geq TP(t_j, T_{rTime}) \mid \forall t_j \in \boldsymbol{T}, \exists\, TP(t_j, T_{rTime}) \quad (19)$$

- Ha a p_i projektnek van előre meghatározott legkorábbi indítási időpontja, akkor annál az időpontnál korábban a projekthez rendelt feladatok egyike sem kezdődhet meg.

$$S_j \geq max_{(p_i,t_j) \in \boldsymbol{TA}}(PP(p_i, P_{rTime})) \mid \forall t_j \in \boldsymbol{T}, \exists\, PP(p_i, P_{rTime}) \quad (20)$$

3.7.2 Erőforrástípus-orientált korlátozások

Az erőforrástípus-orientált korlátozások az egyes erőforrástípusok ütemezési időtől függő terhelésére határoz meg feltételeket.

- Egy erőforrástípus terhelése egyik időpillanatban sem haladhatja meg az erőforrástípus kapacitását.

$$RL_k(\tau) \leq RC_k(\tau) \mid \forall r_k \in \boldsymbol{R}, 0 \leq \tau \leq C_{max} \quad (21)$$

3.8 Célfüggvények

A modell több célfüggvény egyidejű használatát támogatja a legjobb megoldás meghatározásához. A használható célfüggvények száma a modellben nem korlátos, de egy probléma esetén előre meghatározott kell legyen. A modellben megadható a figyelembe vett célfüggvények egyedi prioritása (fontossági értéke), valamint minden célfüggvény esetén az egyedi optimalizálási irány (minimalizálás, maximalizálás).

OFUNCT $\boldsymbol{OFUNCT} = \{f_{PC_{max}}, f_{PDC}, f_{PTCsum}, f_{PTCmax}, \dots, f_{objective}\}$, halmaz jelöli a lehetséges célfüggvények halmazát. Célfüggvényként alkalmazható például a többprojektes ütemezési feladat befejezési határideje ($f_{PC_{max}}$ – *Maximum Project Completion Time*), késedelmes projektek száma (f_{DPC} – *Delayed Project Counter*), összesített projekt késedelmi költség (f_{PTCsum} – *Summary of Projects Tardiness Cost*), legnagyobb projekt késedelmi költség (f_{PTCmax} – *Maximum of Projects Tardiness Cost*) és így tovább. A használható célfüggvények halmaza bővíthető, így új célfüggvény a bemeneti adatokból, és az elsődleges döntési változókból definiálható. Ennek az alfejezetnek egy későbbi részében néhány célfüggvény definícióját példa jelleggel megadom.

$fopt$ $fopt \in \{min, max\}$ függvény jelöli, hogy optimalizáláskor egy adott célfüggvény esetén a célfüggvény szerint számított értéket minimalizálni (*min*), vagy maximalizálni (*max*) szükséges. Ezáltal minden célfüggvény esetén egyedileg meghatározható, hogy két végrehajtható megoldás összehasonlítása esetén egy adott célfüggvény szempontjából melyik megoldást tekintjük jobbnak.

ω ω jelöli egy célfüggvény fontosságát. A fontosságot minden lehetséges célfüggvényre értelmezzük az alábbiak szerint:

$$\begin{cases} 0 < \omega\,, ha\ a\ célfüggvényt\ a\ feladatban\ alkalmazzuk, \\ \omega = 0, egyébként. \end{cases} \quad (22)$$

PO$_i$ $\boldsymbol{PO_i} = \begin{Bmatrix} (\ OFUNCT_1, fopt_1, \omega_1), \\ (\ OFUNCT_2, fopt_2, \omega_2), \\ \dots, \\ (\ OFUNCT_n, fopt_n, \omega_n) \end{Bmatrix}$ halmaz jelöli a többprojektes feladatban szereplő p_i projekt esetén a meghatározott célfüggvényeket, az adott célfüggvény optimalizálási irányát és az adott célfüggvény fontosságát.

A további jelöléseket és alkalmazásokat segítendő az alábbi segédfüggvényeket vezetem be:

$POW_i(OFUNCT)$ függvény határozza meg a p_i projekt $OFUNCT$ célfüggvény szerinti cél fontosságát.

$POD_i(OFUNCT)$ függvény határozza meg a p_i projekt $OFUNCT$ célfüggvény szerinti $fopt$ optimalizálási irányát.

$\boldsymbol{TO_j}$ $\boldsymbol{TO_j} = \begin{Bmatrix} (OFUNCT_1, fopt_1, \omega_1), \\ (OFUNCT_2, fopt_2, \omega_2), \\ \dots, \\ (OFUNCT_n, fopt_n, \omega_n) \end{Bmatrix}$ halmaz jelöli a többprojektes feladatban szereplő t_j feladat esetén a meghatározott célfüggvényeket, az adott célfüggvény optimalizálási irányát és az adott célfüggvény fontosságát.

A további jelöléseket és alkalmazásokat segítendő az alábbi segédfüggvényeket vezetem be:

$TOW_j(OFUNCT)$ függvény határozza meg a t_j feladat $OFUNCT$ célfüggvény szerinti cél fontosságát.

$TOD_j(OFUNCT)$ függvény határozza meg a t_j feladat $OFUNCT$ célfüggvény szerinti $fopt$ optimalizálási irányát.

Az alábbiakban néhány minimalizálandó célfüggvényt mutatok be példaként.

- Többprojektes feladat teljes befejezési időpontja (*multi-project completion time*):
$$f_{PC_{max}} = max_{t_j \in \boldsymbol{T}} (\{C_j\}) = C_{max} \quad (23)$$
- Legnagyobb projekt késés (*maximum project lateness*):
$$f_{PL_{max}} = max_{p_i \in \boldsymbol{P}} (PL(p_i)) \quad (24)$$
- Késő feladatok száma (*total number of late tasks*):
$$f_{TDC} = \sum_{t_j \in \boldsymbol{T}} \lambda_{tdelayed}(t_j) \quad (25)$$
- Legnagyobb feladat csúszás (*maximum task tardiness*):
$$f_{TT_{max}} = max_{t_j \in \boldsymbol{T}} (TT(t_j)) \quad (26)$$
- Feladatok csúszásának összesített ideje (*summarized task tardiness*):
$$f_{TT_{sum}} = \sum_{t_j \in \boldsymbol{T}} (TT(t_j)) \quad (27)$$
- Késő projektek száma (*total number of late projects*):
$$f_{PDC} = \sum_{p_i \in \boldsymbol{P}} \lambda_{pdelayed}(p_i) \quad (28)$$
- Összesített projekt késedelmi költség (*summarized project tardiness cost*):
$$f_{PTCost} = \sum_{p_i \in \boldsymbol{P}} \lambda_{pdelayed}(p_i) * PP(p_i, P_{tCost}) \quad (29)$$

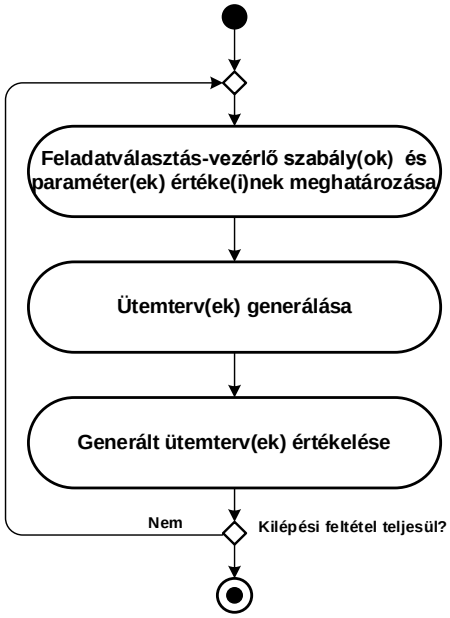
A gyakorlati alkalmazások során számos további célfüggvény használható. A fent bemutatott célfüggvényekhez hasonlóan további célfüggvények definiálhatók.

4 Új módszer a RCMOMPSP feladatok megoldására

Az RCMOMPSP ütemezési feladatok megoldására kidolgoztam egy új megoldási módszert. Ebben a fejezetben először bemutatom az új módszer magas szintű koncepcióját és a megoldási eljárás fő elveit. Ezután részletesen ismertetem az egyes komponensekben alkalmazott algoritmusokat, valamint azokat a kiterjesztéseket, amelyeket az algoritmusokban új elemként alkalmaztam. Az ismertetésben szereplő ábrák esetén az úgynevezett Alapvető Modellezési Koncepciók (*Fundamental Modelling Concept – FMC* [67]) eszközkészletét használom.

4.1 A megoldó módszer magas szintű koncepciója

A kidolgozott módszer több egymást követő lépésből és egy visszacsatolási ágból áll. Az egyes lépések nem függetlenek egymástól; egy adott lépésben hozott döntések és számított eredmények figyelembe vehetőek az azt követő lépésekben. Az egyes lépésekben már ismert algoritmusokat is alkalmazok, melyeket szükség esetén különböző kiterjesztésekkel egészítettem ki a jobb alkalmazhatóság érdekében. A megoldó módszer fő lépéseit mutatja be a 2. ábra.



2. ábra - RCMOMPSP feladatok megoldására javasolt módszer fő lépései

1. lépés: A feladatválasztási szabályok meghatározása. Ezek a szabályok lehetnek manuálisan meghatározottak, vagy keresési eljárással generáltak. Minden feladatválasztási szabálynak lehetnek egyedi prioritásai, valamint további, a szabály által meghatározott vezérlő paraméterei. Ha egy feladatválasztási szabály finomhangolható további paraméterekkel, akkor ebben a lépésben ezeknek az értékeit meghatározzuk.
2. lépés: A definiált feladatválasztási szabályok, feladatválasztást vezérlő paraméterek és a projektütemezési probléma korlátozásainak figyelembevételével megvalósítható ütemterv, vagy ütemtervek generálása.

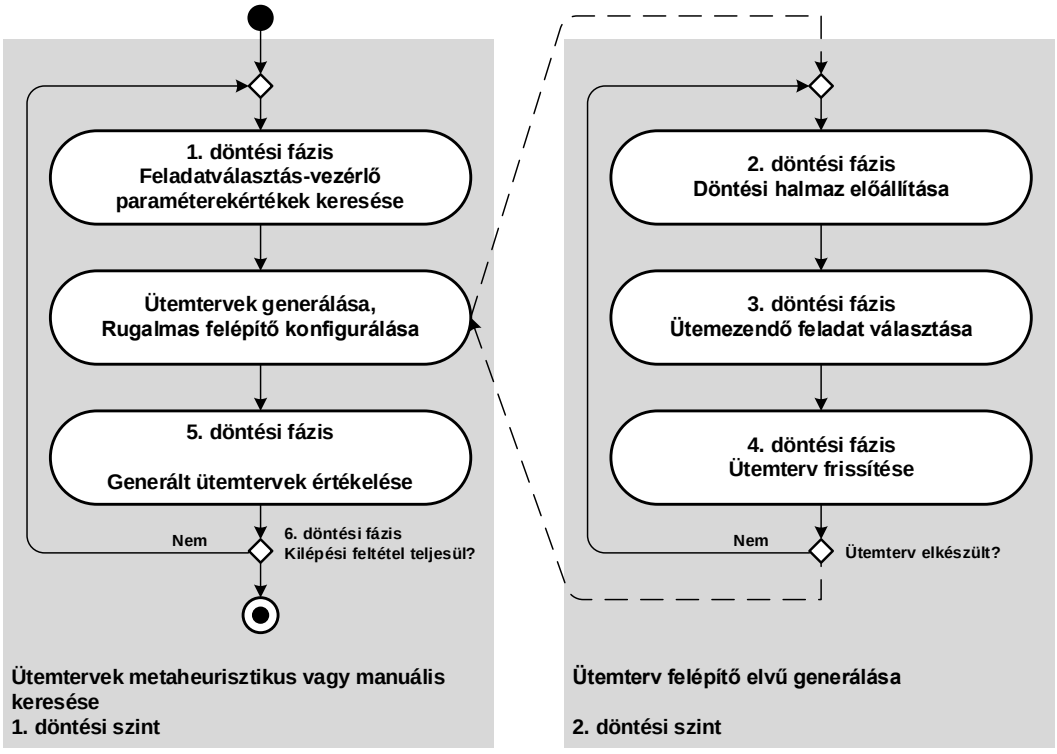
3. lépés: A generált megvalósítható megoldások közül a legjobb megoldások kiválasztása a definiált célfüggvények együttes figyelembevételével. A kiértékelés alapján a keresés iteratív módon ismételhető, vagy az addig azonosított legjobb ütemterv elfogadásra kerülhet.

A megközelítés fontos jellemzője, hogy mindig olyan ütemtervet szolgáltat, amely teljesíti a problémában megadott összes korlátozást. Ha egyetlen végrehajtható ütemterv sem létezik, azt az eljárás kivételként kezeli, és nem megoldható, vagy hibásan definiált ütemezési problémaként határozza meg.

A megoldó módszer végrehajtása során több alkalommal van szükség döntéshozatalra. A döntések lehetnek felhasználói interakció eredményei, de automatikusan generált döntések is szerepet játszhatnak a keresési eljárások során. A fő döntési fázisokat mutatja be a 3. ábra. Az ábrán jól elkülönül a megoldó módszert felépítő két fő egység:

- az ütemterv metaheurisztikus vagy manuális keresése. Keresés során feladatválasztást vezérlő szabályokat választunk és konfigurálunk.
- a végrehajtható ütemterveket felépítő jelleggel generáló eljárás.

Ezeket az egységeket döntési szinteknek nevezem, amelyek több döntési fázist foglalnak magukba. Az első döntési szinten a keresési modul az ütemterv szintjén hoz döntéseket a megtalált ütemterv jelöltekről és további új ütemterv jelöltek generálásának módjáról, a második döntési szint egy konkrét ütemterv generálását befolyásolja.



3. ábra - RCMOMPSP feladatok megoldására javasolt módszer döntési fázisai

Döntési fázisok:

1. döntési fázis: Az alkalmazott feladatválasztási szabályok, azok prioritásai és optimalizálási irányai meghatározásra kerülnek. Ha a feladatválasztási szabály további vezérlő paraméterekkel finomhangolható, akkor a vezérlő paraméternek értéket kell adni. Ezek a választási lépések és értékmeghatározások döntésnek minősülnek. Fontos, hogy a döntés során a már korábban létrehozott ütemtervek figyelembe vehetőek.

2. döntési fázis: Az ütemterv létrehozása során egy felépítő eljárást használok. Az ütemtervet iteratív módon építem fel rendre egy kiválasztott feladat beütemezésével. Minden iterációban egy döntési halmazt állítok elő, amely azokat a feladatokat tartalmazza, amelyek beütemezése a felépítési elv szerint megtehető. A döntési halmaz előállítási módját döntésnek tekintem.

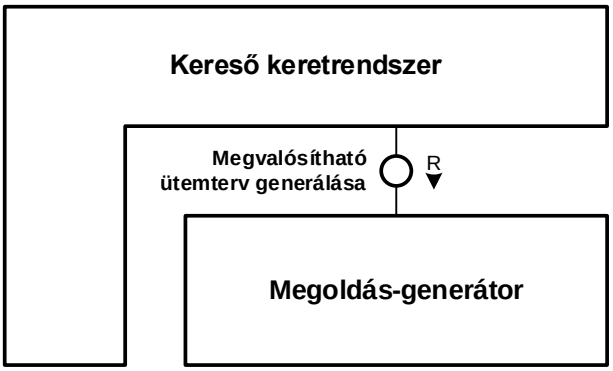
3. döntési fázis: Minden iterációban a feladatválasztási szabályokat együttesen alkalmazva választom ki a következő ütemezendő feladatot a döntési halmazból. A döntést a bemenetként kapott feladatválasztási szabályok, azokhoz tartozó prioritások és optimalizálási irányok, az ütemezési probléma és az elkészült rész-ütemterv alapján kell meghozni. A beütemezendő feladat kiválasztásának módját döntésnek tekintem.

4. döntési fázis: A kiválasztott feladatot a generált részütemtervbe be kell illeszteni. Ezt a beillesztést (beütemezést) döntésnek tekintem. A feladat ütemtervbe illesztése történhet algoritmikusan a lehetséges legkorábbi ütemezési időpont megkeresésével (maximális balra tolással), vagy tartalmazhat további eljárást a szándékos várakoztatás meghatározására. Kutatómunkám során a vizsgált feladatok jellegéből adódóan ebben a döntési fázisban a legkorábbi lehetséges kezdési időpontra illesztés algoritmusát alkalmaztam és a teljes megoldás minőségét a többi döntési fázis módszereire bíztam. A szándékos várakoztatás alkalmazása egy további mély kutatási tématerület, amely több megközelítést is lehetővé tesz. Alkalmazható további kereső-eljárás, vagy heurisztika, hogy egy feladat esetén milyen mértékű szándékos várakoztatás eredményezi a célfüggvények szerinti jobb eredmény elérését.

5. döntési fázis: A generált ütemterv, vagy a generált ütemtervek jóságát az alkalmazott célfüggvények szerint ki kell értékelni. A jóság kiértékelése döntésnek számít, amikor a célfüggvény-rendszer szerint meghatározom, hogy melyik ütemtervet tekintem jobbnak.

6. döntési fázis: Döntést kell hozni arról, hogy befejezhető-e a megoldáskészítési folyamat. Ehhez figyelembe lehet venni az addig elért legjobb ütemtervet, annak jóságát (a célfüggvények értékeit), valamint egyéb algoritmus-végrehajtási mutatókat (pl. végrehajtott iterációk száma, futási idő, javulások jellemzői stb.). A döntést meghozhatja a felhasználó, vagy automatikusan kiértékelhető előre meghatározott kilépési feltételek szerint.

A módszer megvalósítására egy kereső keretrendszert és egy végrehajtható ütemterveket generáló determinisztikus ütemező motor kialakítását javasoltam és alkalmaztam [S8], [S9], [S10], [S12], [S15], [S16], [S18], [S19]. Ez a megközelítés lehetővé teszi a modellezett probléma hatékony megoldását (6. fejezet), ismert ütemezési algoritmusok széleskörű alkalmazását, valamint módot ad az ismert algoritmusok hibrid alkalmazására és különböző továbbfejlesztésére.



4. ábra - Megoldási megközelítés magasszintű koncepciója

A kereső keretrendszerben az első döntési fázisban többcélú, metaheurisztika alapú keresési algoritmusokat alkalmazhatunk. A kereső keretrendszer második lépésében használok egy gyors, determinisztikus algoritmust a garantáltan megvalósítható megoldások generálására. Ezt a koncepcióban megoldás-generátornak neveztem (4. ábra). A megoldás generátorban a generálási sémák működési elvén alapuló, több feladatválasztási szabályt, azok egyedi prioritásait és optimalizálási irányait egyszerre figyelembe vevő algoritmust alkalmazok. Minden esetben garantált, hogy a kiszámított elsődleges döntési változók nem sértik a modellben leírt korlátokat.

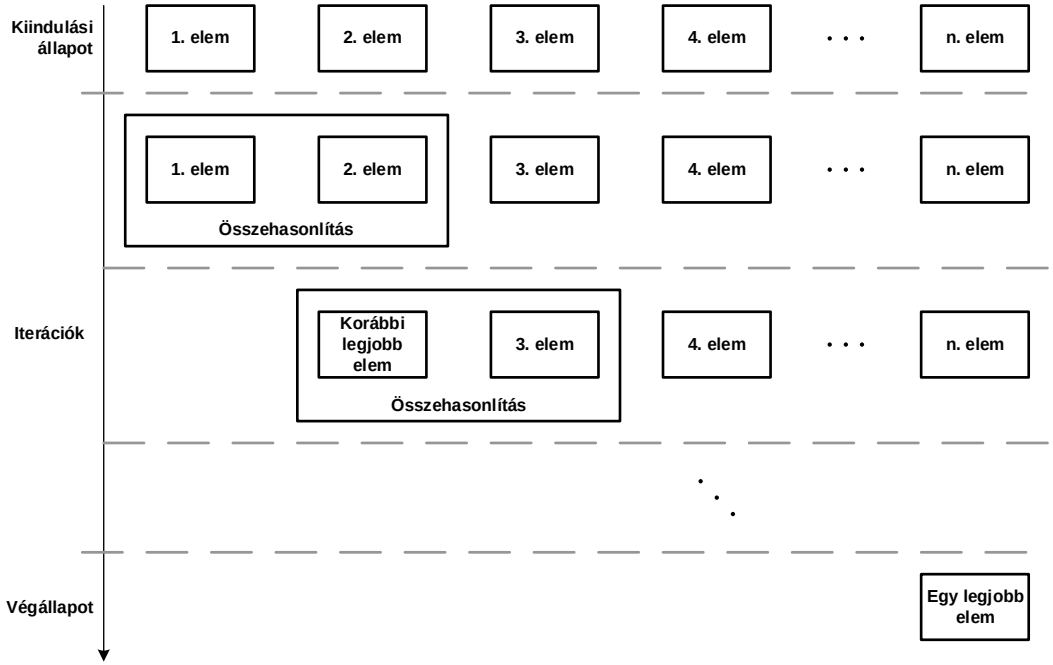
A megoldásom előnye az egyes lépések függetlensége és azok jól meghatározott szerepe, amelynek következtében így azok egymástól függetlenül fejleszthetőek és cserélhetőek. Például a keresési algoritmus esetén a megoldás-generátor által létrehozott megoldásokat különböző módszerekkel és eljárásokkal tudom összehasonlítani, mivel minden lépés jól meghatározott szerepet tölt be a folyamatban. A megközelítésem egyik következménye az alkalmazhatóság rugalmassága, amely számos különböző alkalmazást tesz lehetővé (6. fejezet). Az alábbi alfejezetekben ezt a hibrid ütemezési módszert tovább részletezem és a 7. fejezetben fogalmazom meg a kapcsolódó 2. tézist.

4.2 Többcélú optimalizálás relatív minősítés használatával

A megoldásban használt algoritmusok végrehajtása során többször felmerül az a feladat, hogy két lehetséges elem közül kell kiválasztani egy szempont-rendszer szerint a jobbat. Például a 3. döntési fázisban egy ütemterv generálásakor két lehetséges és végrehajtható feladat közül kell az egyiket ütemezésre kiválasztani. Ugyanez igaz az 5. döntési fázisban is, amikor

két végrehajtható ütemezés közül kell a jobbat meghatározni. Az ilyen típusú döntések kezelésére egy általánosított eljárást fejlesztettem ki, mivel a fent említett két példa formálisan visszavezethető ugyanarra az alapfeladatra. A módszer célja, hogy egy legalább két elemet tartalmazó halmazból a legjobb elemet olyan módon válassza ki, hogy több kiválasztási szempontot egyedi prioritások és a szempontok szerinti optimalizálási irányok mentén egyidőben vegyen figyelembe.

Az összehasonlító algoritmus működési elve az alábbiak szerint fogalmazható meg: A kiválasztási folyamat során mindig a halmaz két eleme kerül kiválasztásra és ezeket az elemeket hasonlítjuk össze egymással. A halmaz elemeit egy tetszőleges sorrendbe rendezzük, majd az első lépésben az első elemet hasonlítjuk össze a második elemmel. Az összehasonlítás eredményeként kiválasztjuk a jobb elemet a következő összehasonlításhoz, és ezt a folyamatot ismételjük, amíg minden elem részt nem vesz legalább egy összehasonlításban. Az eljárás végén egyetlen győztes elem marad, akit a legjobbnak tekintünk az összehasonlítási sorrend szempontjából. Fontos azonban megjegyezni, hogy ha az összehasonlítási módszer nem garantálja a tranzitívitást, akkor a globális legjobb tulajdonságot ez az eljárás nem garantálja, hiszen a végrehajtás eredménye függ az összehasonlítások végrehajtásának kezdeti sorrendjétől. Az eljárás elvét mutatja be az 5. ábra.



5. ábra - A legjobb elem megkeresése egy döntési halmazban előre definiált összehasonlítási sorrend mellett

Két elem összehasonlítására a relatív változásra alapozott összehasonlító módszer [30]. egy továbbfejlesztett matematikai modelljét vezettem be. Legyen adott egy halmaz, melynek elemei egy h valós számot tartalmazó vektorral írhatóak le. Egy gyakorlati példánál ez a h hosszú vektor tartalmazhatja egy adott projekt esetén az egyedi célfüggvények esetében kiszámított értékeket vagy két összehasonlítandó feladat esetén az egyes feladatválasztási

szabályok által adott fontossági értéket. A számítás végrehajtásához további vektorokat vezettem be, az alábbiak szerint:

u $u = (u_1, u_2, \dots, u_g, \dots, u_h), u_g \in \mathbb{R}$; u jelöli egy halmaz egy összehasonlítandó eleme esetén az egyedi szempont alapján számolt valós számokból álló vektort.

w $w = (w_1, w_2, \dots, w_g, \dots, w_h), w_g \in \mathbb{R}$; w jelöli az egyes szempontokhoz tartozó prioritást. Minden w_g egy valós szám, amely meghatározza az u_g érték fontosságát a g-edik szempont esetében.

z $z = (z_1, z_2, \dots, z_g, \dots, z_h), z_g \in \{-1, 1\}$; z jelöli azt a vektort, amely komponensenként határozza meg a szempontok optimalizálási irányát. Abban az esetben, ha a g-edik szempont szerint a kisebb számérték a kedvező, akkor a z_g értéke 1, amennyiben a szempont szerint a nagyobb számérték a kedvező, úgy a z_g értéke -1.

Adott szempont szerint a relatív távolságot a következő függvény adja meg:

$$D : \mathbb{R}^2 \to \mathbb{R},\ D(a,b) = \begin{cases} 0, & ha \max(|a|,|b|) = 0, \\ \frac{b-a}{\max(|a|,|b|)}, & egyébként. \end{cases} \quad (30)$$

A gyakorlati példában az *a* és *b* értékek egy célfüggvény esetén kiszámított értékekre cserélhetőek a távolságfüggvényben. Ebben az esetben a *D* relatív távolságfüggvény értéke kifejezi, hogy második érték (*b*) mennyit változott az első értékhez (*a*) képest egy adott célfüggvény esetén.

Legyen *x* és *y* két *u* típusú vektor. Ez a két vektor a célfüggvények számított értékeit tartalmazzák és két lehetséges, összehasonlítható jelöltet ábrázolnak. Egy valós számot adó *F* függvényt definiálok annak a kifejezésére, hogy az *y* megoldás súlyozott relatív jóságát az *x* megoldáshoz képest kifejezzem:

$$F : u^2 \to \mathbb{R},\ F(x,y) = \sum_{g=1}^{h} (w_g \cdot z_g \cdot D(x_g, y_g)) . \quad (31)$$

Az *F(x,y)* függvény értékét használva ki lehet fejezni az *y* vektor relatív súlyozott jóságát az *x* vektorhoz képest az alábbiak szerint:

- *y* jobb, mint *x*, ha *F(x,y)* értéke kisebb, mint nulla,
- *y* rosszabb, mint *x*, ha *F(x,y)* értéke nagyobb, mint nulla,
- *y* és *x* ugyanolyan jók, ha *F(x,y)* értéke nulla.

A bevezetett relatív minősítési eljárás hatékonyan használható a 3. és 5. döntési fázisban, amikor egy halmaz elemei közül több szempont szerint egyszerre kell kiválasztanunk a legjobb jelöltet, vagy egy új jelölt megoldás relatív javulását kell kiszámítani az aktuális legjobb megoldáshoz viszonyítva.

4.3 Kiterjesztett ütemezés generálási séma

A megoldás generátor komponensének egy lehetséges megvalósítását mutatom be ebben a fejezetben. Az ütemterv generálási séma (*Schedule Generation Scheme – GS*) egy jól ismert típusa az előre definiált, szabály elvű, felépítő jellegű módszereknek. A generálás üres ütemtervből indul ki, és minden iterációjában egy új, megvalósítható részütemezést állít elő úgy, hogy mindig pontosan egy még nem ütemezett feladatot illeszt be az addig elkészített részütemtervbe. Az eljárás addig tart, amíg minden feladat ütemezésre kerül.

Ebben a fejezetben röviden bemutatom a GS eljárást, majd bemutatok egy alkalmazott kiterjesztést, amely lehetővé teszi, hogy az általam javasolt megoldó koncepcióban a generálási séma alkalmazható legyen egy keresési módszerrel vezérelhető formában megoldás generátorként.

4.3.1 Ütemterv generálási sémák

A generálási séma algoritmusa NT számú lépésből áll. Minden egyes m-edik lépésben az alábbi halmazok és vektorok kerülnek determinisztikusan kiszámításra:

$\boldsymbol{TGEN}_m$ $\boldsymbol{TGEN}_m \subseteq \boldsymbol{T}$, $\boldsymbol{TGEN_m}$ jelöli az m-edik iterációban már ütemezett feladatokat.

$\boldsymbol{D}_m$ $\boldsymbol{D}_m$ jelöli azon feladatok döntési halmazát (*decision set* – $\boldsymbol{D}$), amelyek végrehajthatóak az *m*-edik lépésben. $\boldsymbol{D}_m$ halmaz számítási módja függ az alkalmazott generálási séma típusától. Ezeket, mint GS variánsokat később ismertetem.

$SGEN_m$ $SGEN_m = \left(S_1, S_2, \dots, S_j, \dots, S_{NT}\right) |\ S_j$ számolt, ha $t_j \in \mathbf{TGEN}_m$, $SGEN_m$ vektor írja le a már ütemezett feladatok számított kezdési idejét (*Start time generated at stage m)*.

$CGEN_m$ $CGEN_m = \left(C_1, C_2, \dots, C_j, \dots, C_{NT}\right) | C_j$ számolt, ha $t_j \in \mathbf{TGEN}_m$, $CGEN_m$ vektor tartalmazza a már ütemezett feladatok befejezési idejét (*Completion time generated at stage m). C_j* kiszámolható S_j vektorból a (6)-ban definiált összefüggés alapján.

ET_m $ET_m = \left(ET_{t_1}, ET_{t_2}, \dots, ET_{t_j}, \dots, ET_{t_{NT}}\right) | t_j \in \boldsymbol{D_m}$, ET_m vektor tartalmazza a végrehajtható feladatok legkorábbi lehetséges indítási időpontját (*Earliest Times* – *ET*). Egy feladat legkorábbi indítási időpontja nagyobb, vagy egyenlő, mint a feladat megelőző (előfeltételi) feladatainak legnagyobb befejezési időpontja és minimuma az ettől az időponttól meghatározott minden szükséges erőforráskapacitások együttes rendelkezésre állási időpontjai közül. Amennyiben egy $t_j \in \boldsymbol{D_m}$ feladatnak egyéb szigorú indítási megkötései vannak (feladat saját legkorábbi indítási időpontja, feladat végrehajtását magában foglaló projekt legkorábbi indítási időpontja), úgy azt az ET_m számításánál is figyelembe kell venni.

A jelölésrendszert használva a generálási sémák általánosított módszerét mutatja be az 1. algoritmus.

Algoritmus: Ütemterv generálási séma

1: {
2: Bemeneti adatok betöltése;
3: Inicializálás: $m = 0$,
$\boldsymbol{TGEN}_0 = \emptyset$,
$SGEN_0 = (0,0,\dots,0)$,
$CGEN_0 = (0,0,\dots,0)$;
4: Amíg ($\boldsymbol{TGEN}_m <> \boldsymbol{T}$)
5: {
6: $m = m + 1$;
7: Aktualizáld $\boldsymbol{D}_m$;
8: Számold ET_m ;
9: Válassz egy t_m feladatot a $\boldsymbol{D}_m$ döntési halmazból;
10: Frissítsd: $SGEN_m$, $CGEN_m$;
11: $\boldsymbol{TGEN}_m = \boldsymbol{TGEN}_m \cap \{t_m\}$;
12: }
13: Térj vissza: $SGEN_m$, $CGEN_m$;
14: }

1. algoritmus – Ütemterv generálási séma

A generálási sémák variánsainak egyik csoportosítása a soros és párhuzamos generálási sémák szerint történik. A soros generálási sémák a feladat végrehajtási sorrendjén alapuló ütemterv generálási algoritmusokat foglalják magukba, míg a párhuzamosak az ütemezési időpont növelésen alapuló ütemterv generálási algoritmusokat. A két csoport közötti fő eltérés a pszeudokód 7. lépésében feltüntetett döntési halmaz ($\boldsymbol{D_m}$) aktualizálási módszeréből fakad.

- Soros generálási séma esetén (*Serial Generation Scheme – SGS*) minden t_j feladat az algoritmus m-edik lépésében a $\boldsymbol{D_m}$ döntési halmaz eleme, ha a t_j feladat még nem került ütemezésre (azaz nem eleme a $\boldsymbol{TGEN_m}$ halmaznak) és a feladat minden megelőző feladata már ütemezésre került (azaz eleme a $\boldsymbol{TGEN_m}$ halmaznak). A feltétel az alábbi módon formalizálható:

$$\boldsymbol{D_m} = \left(\{\, t_j \,\} \middle| t_j \notin \mathbf{TGEN_m} \wedge (\forall t_{pre} \in \boldsymbol{PRE_j}, t_{pre} \in \mathbf{TGEN_m})\right). \qquad (32)$$

- Párhuzamos generálási séma esetén (*Parallel Generation Scheme – PGS*) minden t_j feladat az algoritmus m-edik lépésében a $\boldsymbol{D_m}$ döntési halmaz eleme, ha az SGS feltétele teljesül és a feladat legkorábbi ütemezési ideje megegyezik az m. lépésben számított ET_m vektor minimum értékével. A feltétel az alábbi módon formalizálható:

$$\boldsymbol{D_m} = \left(\{\, t_j \,\} \middle| t_j \notin \mathbf{TGEN_m} \wedge (\forall t_{pre} \in \boldsymbol{PRE_j}, t_{pre} \in \mathbf{TGEN_m}) \wedge \mathrm{ET}_{t_j} = \min(\mathrm{ET}_m)\right). \qquad (33)$$

A generálási séma döntéshozási alproblémáinak egy másik nagyon fontos eleme, hogy választ kell adni a következő kérdésre: ha döntési halmazban több, mint egy végrehajtható feladat található, akkor hogyan lehet közülük választani?

Az egyik legegyszerűbb módszer, hogy az algoritmus ilyen esetben az m-edik lépésben véletlenszerűen választ egy elemet a $\boldsymbol{D_m}$ halmazból. Ez általában nagyon alacsony hatékonyságot ér el.

A leggyakoribb feladatválasztási módszer a prioritás alapú döntés, melynek során a feladat kiválasztás egy előzetesen meghatározott prioritási szabályon alapul. Ebben az esetben választott feladat az lesz, amelyikhez a kiszámított ütemezési fontosság (*scheduling priority*) - az implementált módszer szerint - a legnagyobb, vagy a legkisebb. Ha több feladat esetén ugyanazt a fontossági értéket kapjuk, akkor vagy véletlenszerűen választunk közülük, vagy egy másodlagos döntési szabályt veszünk figyelembe. Másodlagos szabály lehet például egy előre meghatározott, probléma szinten garantált egyedi feladatsorrend.

Megjegyzés: Az angol szakirodalom a feladatválasztási szabályra a *scheduling priority* vagy a *priority rule* kifejezést használja. Mivel a modellemben a prioritás, vagy súly fogalmát több esetben is használom, ezért az ütemezés generálási sémák esetén a feladatválasztási szabály elnevezést használom, feladatütemezési prioritás és prioritási szabály helyett. Ennek oka, hogy a kiterjesztett modellem (lásd később) a feladatválasztási szabályok szintjén is értelmezi egy szabály fontosságát (prioritását).

A szakirodalomban több feladatválasztási szabályt is megtalálunk. Ezek a feladatválasztási szabályok két fő kategóriába sorolhatóak:

- Statikus feladatválasztási szabályok
  Egy statikus feladatválasztási szabály számított értéke az elkészített részütemtervtől független. Statikus feladatválasztási szabály lehet egy feladat előre definiált konstans jellemzője vagy egy bemeneti adatokból számítható konstans érték. Ezek közé tartozik például egy feladat maximális végrehajtási ideje (az erőforrástípus igények közül a maximális), a feladat befejezési határideje, a feladat egy előre meghatározott konstans jellemzője, a feladatot tartalmazó projekt egy előre meghatározott konstans jellemzője. Emellett statikus feladatválasztási szabályként kezelhetjük a CPM (*Critical Path Method*) módszer szerint számított értékeket, például a feladat erőforrás korlát nélküli ütemterv szerinti legkorábbi befejezési időpontja (*CPM latest finish time – CPM_LFT*).

- Dinamikus feladatválasztási szabályok
  Egy dinamikus feladatválasztási szabály számított értéke függ az elkészített rész-ütemtervtől és az abból származtatott döntési változóktól. A feladatválasztási szabály számított értékét így az algoritmus futásideje során kell kiszámítani, amikor arra kiértékelésnél szükség van. Az angol nyelvű szakirodalomban több dinamikus feladatválasztási szabályt publikáltak a kutatók, melyeket *dynamic priority rules*-nak neveztek el. Például egy dinamikus feladatválasztási szabály lehet a minimális tartalékidő (*minimal slack – MS*).

A generálási sémák variánsa és a generáláskor alkalmazott feladatválasztási szabály tetszőlegesen kombinálható, hogy egy ütemezési célt megvalósítsunk. Ugyanakkor, ha sok célfüggvényt kell egyidejűleg figyelembe venni, akkor a gyakorlatban a feladatválasztási szabályok alkalmazhatósága korlátozott. Ennek oka, hogy nehéz az olyan feladatválasztási szabály meghatározása, amely jól közelít a kitűzött célokhoz. További kihívást jelent, hogy az ütemező generálási eljárás rugalmassága korlátozott, és a célfüggvények kis mértékű változása az ütemterv generálási eljárás jelentős módosításával járhat. A vázolt probléma megoldására egy kiterjesztett ütemterv generálási eljárást javaslok.

4.3.2 Rugalmasan vezérelhető, több prioritásos generálási séma (FCMPGS)

Megközelítésemben egy kiterjesztett ütemterv generálási eljárásra teszek javaslatot, amelynek egy további minősítő jellemzője a generálási séma rugalmas konfigurálhatósága és paraméteres vezérelhetősége. Az új minősítő jellemző előnye, hogy lehetővé teszi a felépítő jellegű stratégiák alkalmazását olyan többcélú, az alkalmazott célfüggvények szempontjából merőben eltérő esetekben is, amelyre a korábbi feladatválasztási szabályok nem voltak hatékonyan alkalmazhatóak.

Az így kiterjesztett ütemterv generálási sémát rugalmasan vezérelhető, több prioritásos generálási sémának (*Flexibly Controllable Many-Priority Generation Scheme – FCMPGS*) neveztem el.

4.3.2.1 FCMPGS vezérlési paraméterei

Az FCMPGS futásának konfigurálására és a működés finomhangolására több módszer áll rendelkezésre. Megadható, hogy a generálási séma milyen variáns alapján állítsa elő a döntési halmazt, amelyből a következő ütemezendő feladatot kiválasztja. Megadható, hogy milyen feladatválasztási szabályok milyen prioritásokkal és optimalizálási irányokkal számítsanak, amikor az algoritmus a döntési halmazból a következő ütemezendő feladatot választja. Megadható, hogy mi történjen akkor, ha a feladatválasztási szabályok alkalmazása után is több feladat is ugyanolyan ütemezési fontosságot kapott.

Az FCMPGS számára meghatározható ilyen módon konfigurálható és finomhangolható paraméterek és lehetséges értékkészletük az alábbiak szerint jelölt:

ψ_m — $\psi_m \in \{\psi_s, \psi_p, \psi_d\}$, ψ_m jelöli az alkalmazott generálási séma variánsát. A variáns lehet soros generálási séma (*serial generation scheme* - ψ_s), párhuzamos generálási séma (*parallell generation scheme* - ψ_p), vagy direkt generálási séma (*direct generation scheme* - ψ_d). Direkt generálási séma esetén a döntési halmaz egy előre meghatározott sorrend szerint adott.

χ — $\chi \in \{\chi_r, \chi_{tid}\}$, jelöli a következő ütemezendő feladat kiválasztási módszerét, amit abban az esetben kell alkalmazni, ha a döntési halmaz a számított prioritások szerint több, mint egy végrehajtásra választható feladatot tartalmazna. Ez lehet véletlenszerű választás (*random* - χ_r), vagy a feladatok egy globális, egyedi

azonosító szerinti explicit sorrendje (*task global identifier* - χ_{tid}). A globális egyedi sorrend alapján az FCMPGS eljárás biztosan determinisztikus marad.

$GSTP$ $GSTP = (GSTP_1, GSTP_2, \dots, GSTP_{NTP})$, $GSTP$ vektor jelöli a generálási séma által alkalmazott feladatválasztási szabályokat, amelyben NTP (*Number of Task Priority*) jelöli az alkalmazott feladatválasztási szabályok számát.

δ $\delta = (\delta_1, \delta_2, \dots, \delta_{NTP})$, δ jelöli a megadott fontossági prioritásokat a feladatválasztási szabályok esetében. A nagyobb prioritású feladatválasztási szabály nagyobb hatást gyakorol a döntésben.

ϱ $\varrho = (\varrho_1, \varrho_2, \dots, \varrho_{NTP})$, ϱ jelöli az egyes feladatválasztási szabályok egyedi optimalizálási irányát, amely kifejezi, hogy a nagyobb vagy a kisebb érték kedvezőbb.

4.3.2.2 Ütemterv generálás többcélú vezérelhetősége

A kiterjesztett megközelítés figyelembe veszi a feladatválasztási szabályok rugalmas definiálásának lehetőségét. Minden feladatválasztási szabály egyedi döntési szempontot reprezentál a saját számított értékével, az optimalizálási irányával és egyedi fontosságával (prioritásával). Például egy statikus feladatválasztási szabály esetén két feladat közül azt kell korábban ütemeznünk, amelyik esetén a számított érték nagyobb. Egy másik, dinamikus feladatválasztási szabálynál előfordulhat, hogy egy feladatot hamarabb kell választanunk a döntési halmazból, ha a dinamikusan számított értéke alacsonyabb. Ha ezt a két szabályt együtt akarjuk figyelembe venni a döntés meghozatalában, akkor a szabályokhoz rendelt prioritások határozzák meg a figyelembevétel mértékét.

A $\boldsymbol{D_m}$ döntési halmazból a következő ütemezendő feladat választása szükségessé teszi, hogy a különböző alkalmazható feladatválasztási szabályok, valamint azoknak a feladat választásban számított prioritásai és optimalizálási irányai figyelembevételre kerüljenek. Ennek megoldására a korábban bevezetett *F* függvény (28) alkalmazható. Ehhez szükséges, hogy a generálási séma adott iterációjában ismerjük egy t_j feladat esetén az egyes feladatválasztási szabályok számolt értékét. Ezt az alábbiak szerint jelölöm.

$v(t_j, m)$ $v(t_j, m) = \left(v(t_j, m)_1, v(t_j, m)_2, \dots, v(t_j, m)_{NTP} \right)$ jelöli az ütemtervgenerálás m-edik lépésében a t_j feladat esetén a feladatválasztási szabályok által számolt értékeket.

A megadott jelölésekkel a relatív összehasonlítás módszere a több prioritásos feladat választási lépésben az alábbi módon alkalmazható:

1. A feladatválasztási szabály megfeleltethető a szempontnak. Az összehasonlítás alapjául szolgáló számszerű értékek az ütemezés m-edik lépésében a t_j feladat esetén az egyes feladatválasztási szabályok számított értékei:

$$u(t_j) = v(t_j, m) = \left(v(t_j, m)_1, v(t_j, m)_2, \dots, v(t_j, m)_{NTP} \right). \qquad (34)$$

2. Az egyes feladatválasztási szabályokhoz rendelt fontosság (prioritás) meghatározza a szempont fontosságát. A δ vector NTP számú értéket tartalmaz. Minden érték egy valós szám, amely kifejezi a hozzá tartozó feladatválasztási szabály fontosságát a relatív összehasonlítás során.

$$w = \delta = (\delta_1, \delta_2, \dots, \delta_{NTP}) \quad (35)$$

3. A feladatválasztási szabályhoz rendelt egyedi optimalizálási irányok együtt kifejezik a kiválasztási döntéshozatal optimalizálási irányát.

$$z = \varrho = (\varrho_1, \varrho_2, \dots, \varrho_{NTP}) \quad (36)$$

A rugalmasan vezérelhető több prioritásos generálási séma mindig alkalmazható tradicionális, ütemezés felépítő módszerként is. Ha egyszerre csak egy feladatválasztási szabályt alkalmazunk és a döntési halmazból választanunk kell a szabály alapján, akkor eredményül a klasszikus, egy feladatválasztási elvet használó generálási sémát kapjuk. A párhuzamos vagy soros generálási sémák alkalmazhatósága hasonlóan egy bemeneti paraméterrel vezérelhető, hiszen működésükben különbség csak az ütemezésre kiválasztható feladatok ($\boldsymbol{D_m}$) meghatározásában van, amelyet nem befolyásol a tényleges feladatválasztási lépés. Következésként a kiterjesztett módszerem legalább olyan hatékony, mint a tradicionális, egy feladatválasztási szabályon alapuló, generált ütemtervet felépítő módszerek.

A javasolt megközelítésem fő előnye abban rejlik, hogy több feladatválasztási szabályt alkalmazva gyorsan és determinisztikusan létre tudunk hozni egy végrehajtható ütemtervet. Az eljárás rendkívül rugalmas és nem korlátozza az alkalmazható feladatválasztási szabályok típusát és számát. Emellett a felhasználó vagy integráló komponens által megadott fontossági értékek segítségével az ütemterv kialakítási eljárás finomhangolható a modell megváltoztatása nélkül. Ezzel a kiterjesztéssel az FCMPGS módszer hatékonyan alkalmazható más megoldó algoritmusok részeként is, így alkotva hibrid megoldási megközelítést. Az értekezés 6. fejezetében különböző példákat mutatok be az új feladatkiválasztási módszer használatára és a 7. fejezetben fogalmazom meg a kapcsolódó 3. tézist.

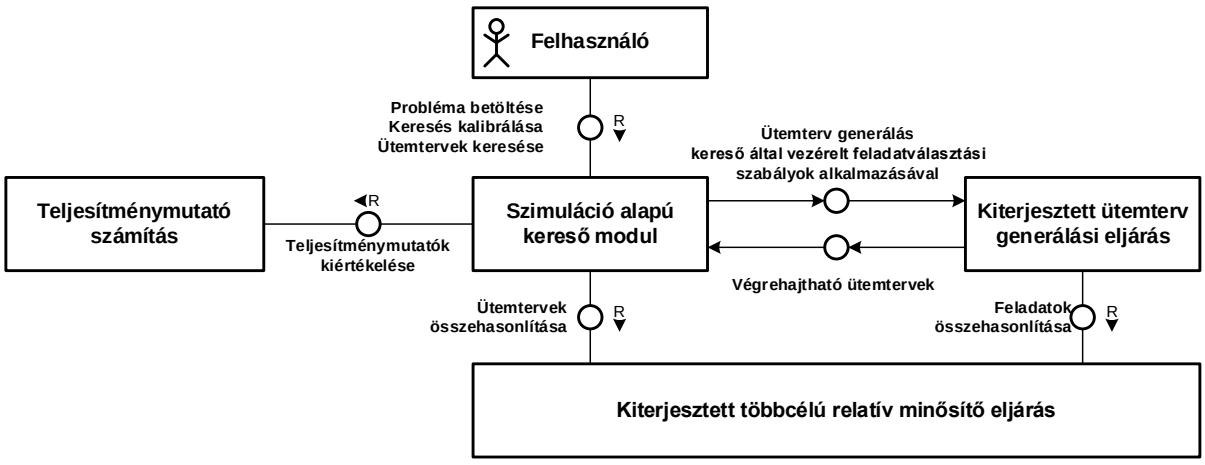
4.4 Hibrid módszer alkalmazása az RCMOMPSP probléma megoldására

Az előző 4.3.2 alfejezetben bemutatott FCMPGS generálási séma széles körben használható RCPSP, RCMPSP és RCMOMPSP problémák végrehajtható ütemterveinek gyors előállítására. Számtalan kutatási munka eredménye igazolja azonban, hogy a hagyományos felépítési elvű generálási sémák által előállított ütemterv minősége elmarad a metaheurisztikus keresők által létrehozott ütemtervek minőségétől. Hagyatkozva a kutatási munkák tapasztalataira, valamint arra a megfigyelésre, hogy az FCMPGS módszer magában rejti a két eljárás előnyeinek egyidejű használatát, javaslatot teszek egy új, hibrid megközelítésre.

Mivel az RCMOMPSP probléma az RCPSP probléma valós kiterjesztése, ezért az RCMOMPSP probléma is az NP-nehéz feladatok körébe tartozik. Gyakorlati problémák esetén lehetetlen az összes végrehajtható ütemezést összegyűjteni, azokat a célfüggvények szerint kiértékelni és összehasonlítás után a legjobbakat kiválasztani, mivel a probléma keresési tere és a számításhoz szükséges idő elfogadhatatlanul nagy a kombinatorikus robbanás miatt. Ez azt

eredményezi, hogy a kis méretű problémáktól eltekintve a nyers erőn (*brute force*) alapuló megközelítések nem működnek. A kereső-alapú ütemezők kompromisszumos megoldást kínálnak, vagyis a teljes kereséstől eltérően jelentősen kisebb számú lehetséges ütemtervet hoznak létre és vizsgálnak meg. Ugyanakkor nem garantált, hogy a megtalált ütemterv a globálisan lehetséges optimumnak felel meg. Heurisztikus algoritmusok elfogadható megközelítésnek számítanak azokban az esetekben, ha az esetek túlnyomó többségében az ütemező által létrehozott ütemterv megegyezik az optimummal vagy optimumhoz elfogadhatóan közeli megoldást ad. Az optimális ütemtervhez elfogadhatóan közeli ütemterv megítélése mindig az alkalmazás és az adott terület igényeitől függ, nincs rá egzakt meghatározás.

Az RCMOMPSP probléma megoldására a 4.1 fejezetben vázolt hibrid módszert dolgoztam ki. Az újszerű megközelítés lényege, hogy egy kiterjesztett, többcélú kereső algoritmus iteratív módon úgynevezett jelölt vezérlő értékeket (*Candidate Control Value – CCV*) állít elő egy beépített FCMPGS eljárás számára. Az FCMPGS a kereső számára egy konstruktív szimulációs komponensnek tekinthető, amely determinisztikusan képes előállítani egy végrehajtható ütemtervet. A kereső algoritmusban a CCV értékei a szabad döntési változókat képviselik, amelyek a felépítő algoritmusnak átadva a beütemezendő következő feladat kiválasztási fázisban releváns feladatválasztási szempont értékeivé válnak. Az FCMPGS számára a kereső által adott feladatfontosság értékek csupán bemeneti adatok, amelyet a kereső, vagy a felhasználó adott meg. A tisztán kereső modul által vezérelt konfiguráció szerint a kereső által generált CCV lesz az egyetlen döntési szempont, amelyet a felépítő algoritmus a feladatválasztás során figyelembe vesz. Más konfigurációban, amikor a kereső által generált CCV értékeket alkalmazó feladatválasztási szabály mellett egyéb feladatválasztási szabályt is kell alkalmazni a következő ütemezendő feladat választásakor, akkor az FCMPGS a több prioritásos relatív minősítési eljárását alkalmazza. Ilyen módon a kereső modul működése kombinálható más feladatválasztási szabályokkal, például a felhasználó bemeneti konfigurációjával vagy egy másik kereső eljárással. A hibrid megoldó koncepcionális ábráját mutatja be a 6. ábra.



6. ábra - Hibrid ütemező koncepcionális felépítése

4.4.1 A generálási séma vezérlése

Bevezetek egy új feladatjellemzőt T_{cPrio} jelöléssel, amely egy szabadon választott valós szám és a kereső modul által kerül meghatározásra minden ütemezendő feladat (*task*) számára.

T_{cPrio} $T_{cPrio} \in \mathbb{R}$, T_{cPrio} jelöli az új, feladathoz rendelt keresési prioritást

Amennyiben a GSTP vektor csupán a T_{cPrio} értéken alapuló, hasonlóan T_{cPrio}-nak nevezett feladatválasztási elvet veszi figyelembe, akkor az FCMPGS futási eredménye csak a kereső modul által generált CCV értékektől függ. Az FCMPGS minden esetben a kereső által előállított prioritási értékek alapján választ a döntési halmazból, és a generált ütemterv minőségi jellemzői kizárólag a szimuláció alapú keresőtől függenek. Ezáltal kijelenthető, hogy a jelölt ütemterv végrehajtható és reprezentálja a CCV paraméterek jóságát.

Ennek következtében a CCV vektor maga reprezentálhatja az ütemtervet. Ezen megközelítésnek köszönhetően több metaheurisztika alapú kereső algoritmus is alkalmazhatóvá válik a kereső modulban. Amennyiben minden keresési vezérlő feladatválasztási érték egyedi (nincs két egyforma szám a CCV vektorban), akkor a keresési modul permutáció alapú keresési feladattá redukálható.

4.4.2 Ütemtervek többcélú relatív minősítésen alapuló összehasonlítása

A generált végrehajtható ütemtervek minőségének mérését egy külön modul végzi, a teljesítménymutató számítási logika (*Key Performance Indicator Calculation – KPI Calculator*). Ez a modul megkapja a kereső modultól a generált ütemterveket, majd minden ütemtervre kiszámítja az előre meghatározott célfüggvények értékét. Egy adott célfüggvény értéke függ a modellezett problémától és a generált ütemtervtől. A kiszámított teljesítménymutatókat az alábbi módon jelölöm:

φ_e $\varphi_e = (\varphi_{e,1}, \varphi_{e,2}, \dots, \varphi_{e,NE})$, φ_i jelöli a célfüggvények számított értékeit. *NE* számú célfüggvényt alkalmazunk. $\varphi_{e,n}$ érték lehet többek között a p_i projekt esetén a problémában meghatározott $\boldsymbol{PO}_i$ célfüggvény szerint számított érték, t_j feladat feladat esetén a problémában meghatározott $\boldsymbol{TO}_j$ célfüggvény szerint számított érték, alkalmazhatunk erőforrásra számított célfüggvény értékeket és így tovább.

A kereső algoritmus futása során több célfüggvényt lehet együttesen alkalmazni annak érdekében, hogy a legjobb jelöltet kiválasszuk a végrehajtható ütemtervek közül. Ez a folyamat megfelel az áttekintőben ismertetett 5. döntési fázisnak. A döntéshez – a korábbiakhoz hasonlóan – többcélú, relatív minősítés alapú modellt használok az alábbiak szerint.

SCH_1 és SCH_2 legyen két különböző végrehajtható ütemterv. Az összehasonlításhoz létre szeretnénk hozni két vektort. Legyen x és y két u típusú vector. Ezek a vektorok úgy hozhatóak létre, hogy az összes célfüggvény szerinti számított értéket indexpáronként egymásnak megfeleltetve beszúrjuk. A számított értékek az x vektorban megfelelnek az SCH_1 ütemtervnek, míg az értékek az y vektorban az SCH_2 ütemtervnek. Optimalizálási

szempontból x és y képviselik a két jelölt ütemtervet. Egy adott SCH_1 és SCH_2ütemterv esetén a célfüggvények számított értékeit reprezentálja a $\varphi_{e(SCH_1)}$ és $\varphi_{e(SCH_2)}$.

Követve ezt a felépítési szabályt hasonlóan létrehozhatunk egy *w* vektort, amely az egyedi fontossági súlyokat tartalmazza az egyedi célfüggvények szerint. Az előre meghatározott egyedi fontossági súlyok, melyek az alkalmazott célfüggvények prioritásai a *w* vektor részét képezik.

Végül egy *z* vektort is létrehozhatunk az egyedi célfüggvények egyedi optimalizálási irányaiból. Az előzőekben bemutatott elv szerint az optimalizálási irányok a *z* vektor részét képezik. A különböző vektorokban feltüntetett, de egymáshoz kapcsolódó értékeket a közös indexük (pozíciójuk) köti össze.

Ezt a leképzési eljárást használva a relatív összehasonlításnál bemutatott *F* függvény ebben az esetben is alkalmazhatóvá válik. Ezáltal a többcélú relatív minősítő eljárás effektív módon oldja meg a kereső szintjén két végrehajtható megoldás összehasonlítását az 5. fázisban.

4.5 Példák a hibrid megközelítésben kombinált algoritmusokra

Ebben az alfejezetben három konkrét hibrid implementált módszert fogok bemutatni, amelyekben metaheurisztikus keresőt kombinálok az FCMPGS-el. Az első példában egy lokális kereső algoritmus vezérli a felépítő algoritmust. A második példában egy kiterjesztett evolúciós algoritmus szolgáltatja a vezérlő prioritásokat a felépítő algoritmus számára. Végül a harmadik példa egy új, kombinált evolúciós és lokális keresést kapcsol össze a felépítő algoritmussal.

Az általam fejlesztett evolúciós algoritmus egy meglévő kereső eljárás kiterjesztésének tekinthető és bizonyos ismert keresési feladatokban hatékonyabb megoldást tud nyújtani, mint az eredeti eljárás. Ezt az értekezésben külön eredményként tüntetem fel a 6.1.2 alfejezetben. A kapcsolódó 5. tézist a 7. fejezetben fogalmazom meg.

A populáció-alapú és lokális keresési elvek kombinálásával létrehozott hibrid kereső az FCMPGS felépítő egy speciális változatával együttműködve bizonyult a teszteken a leghatékonyabb megoldómódszer-változatnak (6. fejezet).

Fontos megjegyezni, hogy a kifejlesztett megoldási módszer alkalmazható bármilyen ismert metaheurisztikus kereső esetén az itt bemutatott példa implementációk mintája alapján.

4.5.1 Lokális szomszédsági keresés

A bemutatásra kerülő lokális szomszédsági kereső algoritmus egy egyszerű, mintavételes, szomszédsági, kvázi-gradiens elvű kereső algoritmus. A kereső modul két bemeneti vezérlő paraméterrel rendelkezik:

$nsmax$ $nsmax \in \mathbb{Z}^+$, egy keresési iterációban generált lehetséges szomszédok száma.

$nimax$ $nimax \in \mathbb{Z}^+$, a keresési iterációk maximális száma.

Algoritmus: Lokális szomszédsági keresés FCMPGS felépítő használatával

1: **{**
2: Bemeneti adatok betöltése;
3: Kezdeti T_{cPrio} prioritások generálása minden feladatra;
4: FCMPGS ütemterv generálási séma végrehajtása;
5: Teljesítménymutatók kiértékelése és a legjobb megoldás beállítása ($bestSolution$);
6: Ismételd (i=0; i < $nimax$; i++)
7: {
8: Ismételd (n=0; n < $nsmax$; n++)
9: {
10: T_{cPrio} módosítása $bestSolution$ két tetszőlegesen választott feladata számára, a kiválasztott módosítási operátor alapján;
11: FCMPGS ütemterv generálási séma végrehajtása;
12: Teljesítménymutatók kiértékelése a megoldásjelölt ütemtervek esetén, a $candidateSolution$ halmazhoz adás;
13: }
14: $bestSolution$ és a $candidateSolution$ összehasonlítása az F függvénnyel;
15: $bestSolution$ frissítése;
16: }
17: Térj vissza a $bestSolution$ ütemtervvel;
18: }

2. algoritmus - Lokális szomszédsági keresés FCMPGS felépítő használatával

A lokális szomszédsági kereső algoritmus egy véletlenszerűen generált ütemtervből indul ki. Ehhez a szimulátorként használt generálási séma számára inicializáláskor egy véletlenszerűen generált számot alkalmaz (T_{cPrio}), és garantálja, hogy egy T_{cPrio} érték csak egyszer kerül felhasználásra. Ez legegyszerűbben úgy érhető el, hogy a feladatok számának megfelelő egész számokat rendel véletlenszerűen az egyes feladatokhoz. A generált T_{cPrio} értékekkel rendelkező megoldásjelölt az FCMPGS generálási sémát használva egy végrehajtható ütemtervet generál. A kezdeti megoldást tekintjük a legjobb megoldásnak ($bestSolution$). Az eljárás minden iterációban *nsmax* számú szomszédot generál az ismert legjobb megoldás alapján, mely során az alábbi módosítási operációk valamelyikét alkalmazza:

- Egy véletlenszerűen választott feladat ütemezési prioritását egy vele szomszédos feladattal megcseréli
- Két véletlenszerűen választott feladat ütemezési prioritását megcseréli

A generált szomszédos megoldás jelölteket az FCMPGS alkalmazásával szimulálja, amely felépítő eljárás a már módosított T_{cPrio} értékeket veszi figyelembe. A generált ütemtervek az FCMPGS felépítési elve miatt végrehajthatóak. A generált ütemterveket a célfüggvények szerint kiértékeljük, majd a többcélú, relatív minősítő összehasonlító eljárással az új legjobb megoldás kerül kiválasztásra. Az algoritmus az előre meghatározott maximális

iterációszámig ismétlődik. További szofisztikált kilépési feltételek is alkalmazhatóak, úgymint korlát az olyan eltelt iterációk számára, amikor már nem talált jobb megoldást a kereső.

Annak ellenére, hogy a kereső eljárás egyszerű, jól mutatja be, hogy a metaheurisztikus kereső eljárás során az FCMPGS algoritmust szimulátorként használva a feladat egy permutáció alapú kereső feladatra redukálható. Ennek oka, hogy a feladatokhoz rendelt T_{cPrio} értékek alapján a feladatok egy egyértelmű sorrendbe rendezhetőek és a módosító operátorok csupán a feladatok sorrendjét módosítják.

4.5.2 Kiterjesztett többcélú Jaya evolúciós algoritmus

Megvizsgáltam egy populáció alapú, evolúciós algoritmus alkalmazhatóságát. A vizsgálat során a populáció egyedeinek kiértékeléséhez az FCMPGS eljárást használtam konstruktív szimulációs modulként. A vizsgálat során egy ismert evolúciós algoritmusból indultam ki, amelyhez új kiterjesztést vezettem be. Az alkalmazott kiterjesztéssel néhány ütemezési probléma típus esetén az eredeti evolúciós algoritmustól jobb teljesítményű eljárást kaptam.

A vizsgált algoritmus a Jaya algoritmus, ami egy viszonylag új metaheurisztikus algoritmus, amelyet Rao javasolt 2016-ban [28] korlátos és korlát nélküli optimalizálási problémák megoldására. A Jaya algoritmus egyszerű, de nagyon hatékony optimalizálási technika, amelyet kutatók sikeresen alkalmaztak több tudományos és mérnöki területen.

Az eredeti JAYA algoritmus egyetlen $f(x)$ célfüggvényt alkalmaz. A keresés minden *i*-edik iterációja *m* különböző döntési változót ($j = 1,2, \dots, m$) és *n* darab megoldásjelöltet ($k = 1,2, \dots, n$) alkalmaz. A *best* szimbólum jelöli az ismert legjobb megoldásjelöltet, amely esetén az $f(x)$ függvény értéke a legkisebb a populáció megoldásjelöltjei közül. Hasonlóan a *worst* szimbólum jelöli azt a megoldásjelöltet, amely esetén az $f(x)$ függvény értéke a legnagyobb a populáció összes megoldásjelöltje közül. $X_{j,k,i}$ jelöli a *j*-edik döntési változót a *k*-adik megoldásjelölt esetén az *i*-edik iterációs során, míg $X'_{j,k,i}$ jelüli az $X_{j,k,i}$ módosított értékét. Az eredeti jelölésrendszert használva a döntési változó értéke az i. iterációban az alábbi módon kerül módosításra:

$$X'_{j,k,i} = X_{j,k,i} + r_{1,j,i}\left(X_{j,best,i} - \left|X_{j,k,i}\right|\right) - r_{2,j,i}\left(X_{j,worst,i} - \left|X_{j,k,i}\right|\right) \qquad (37)$$

ahol $r_{1,j,i}$ és $r_{2,j,i}$ két véletlenszerűen generált valós szám a [0, 1] intervallumból a *j*-edik döntési változó esetén az i -edig iterációban. Ha a módosított $X'_{j,k,i}$ esetén a megoldásjelölt számított célfüggvény értéke jobb, mint $X_{j,k,i}$ esetén, akkor $X'_{j,k,i}$ értéket fogadjuk el, egyébként a módosított jelölt elvetésre kerül. A döntési változók elfogadott értékei alkotják a következő evolúciós iteráció kezdeti populációját.

Az eredeti Jaya algoritmust a többcélú célfüggvény-rendszer alkalmazásával terjesztettem ki. Az új eljárást kiterjesztett többcélú Jaya algoritmusnak neveztem el (*Advanced Multi-Objective Jaya Algorithm – AMOJ*). A kiterjesztés fő jellemzői:

- Lehetővé teszi több célfüggvény egyidejű használatát, egyedi optimalizálási irányokkal és prioritásokkal.
- Az új populáció generálása során a döntési változók értékeinek módosítása egy új operáció használatával történik meg.

Az eredeti JAYA algoritmus jelöléseit használva az AMOJ algoritmus módosító operációja a következő:

$$X'_{j,k,i} = X_{j,k,i} + r_{1,j,k,i}\left(X_{j,best,i} - X_{j,k,i}\right) - r_{2,j,k,i}\left(X_{j,worst,i} - X_{j,k,i}\right) + r_{3,j,k,i}\left(X_{j,rand1,i} - X_{j,k,i}\right) - r_{4,j,k,i}\left(X_{j,rand2,i} - X_{j,k,i}\right) \quad (38)$$

ahol az $r_{1,j,k,i}$, $r_{2,j,k,i}$, $r_{3,j,k,i}$ és $r_{4,j,k,i}$ véletlenszerűen generált valós számok a [0,1] intervallumból a *j*-edik döntési változó esetén, a k-adik megoldásjelölt esetén az *i* -edig iterációban. Továbbá $rand1$ és $rand2$ jelöli a populáció két véletlenül választott megoldásjelöltjét. Ez az új integrált rekombinációs és mutációs operátor adja a 7. fejezetben megfogalmazott 5. tézis kulcselemét. A kapcsolódó tesztek eredménye a 6.2.1 alfejezetben érhető el.

Az AMOJ algoritmus két bemeneti vezérlő paraméterrel rendelkezik:

$gspopsize$ $gspopsize \in \mathbb{Z}^+$, egy keresési iterációban generált populáció mérete

$gsimax$ $gsimax \in \mathbb{Z}^+$, a keresési iterációk maximális száma

Az RCMOMPSP probléma megoldására az AMOJ algoritmus keresési eljárásként alkalmazható, amely során a populáció egyedei határozzák meg az FCMPGS algoritmus számára a kereső által meghatározott vezérlési paraméterek értékét. Ebben az alkalmazásban az $X_{j,k,i}$ döntési változó azt jelenti, hogy a j-edik feladat esetén a k-adik megoldásjelölt esetén az i-edik iterációban mi a T_{cPrio} értéke.

Az AMOJ algoritmus által létrehozott megoldásjelöltek összehasonlítására a többcélú relatív minősítési eljárás alkalmazható (*F* függvény).

Az AMOJ módszert ismerteti a 3. algoritmus.

	Algoritmus: AMOJ
1:	{
2:	Bemeneti adatok betöltése;
3:	Ismételd (pop=0; pop < $gspopsize$; pop++)
4:	{
5:	Generálj kezdő T_{cPrio} értékeket minden feladathoz az egyed számára;
6:	Hajtsd végre az FCMPGS algoritmust;
7:	Értékeld ki a teljesítménymutatókat;
8:	Add az egyedet a populációhoz;
9:	}

10: F függvény használatával határozd meg a legjobb és legrosszabb egyedet;
11: Ismételd (gsidx=0; gsidx < $gsimax$; gsidx++)
12: {
13: Ismételd (pop=0; pop < $gspopsize$; pop++)
14: {
15: Módosítsd a populáció pop tagját az előző generáció legjobb és legrosszabb egyede alapján (38) képlet szerint;
16: Hajtsd végre az FCMPGS algoritmust;
17: Értékeld ki a teljesítménymutatókat az új egyed esetén;
18: Az új generáció egyedét hasonlítsd össze az előző generáció egyedével és a jobbat add hozzá az új populációhoz;
19: }
20: F függvény használatával határozd meg a legjobb és legrosszabb egyedet;
21: }
22: Térj vissza a populáció legjobb egyedével;
23: }

3. algoritmus - Advanced Multi-Objective Jaya (AMOJ)

Az új integrált rekombinációs és mutációs operátort is magába foglaló AMOJ algoritmus futási eredményei a 6.1.2 fejezetben szerepelnek. Az AMOJ eljárás sikeres alkalmazása bizonyítja, hogy az RCMOMPSP probléma kezelhető bármely populáció alapú optimalizálási metaheurisztikus keresővel, amely a keresési folyamat során az FCMPGS algoritmust használja konstruktív szimulátorként. Ebben az esetben a populáció egyedeinek egyedi döntési változói az FCMPGS algoritmus feladatválasztási lépésére képezhetők le.

4.5.3 Rekombináció nélküli, kiterjesztett evolúciós keresőalgoritmus

A Jaya algoritmus megismerése és alkalmazása után egy új algoritmust dolgoztam ki az ütemezendő feladatok prioritásának meghatározására. Az új algoritmus a JAYA algoritmus alapötletét kombinálja lokális kereséssel. Ennek eredményeként egy kombinált kereső eljárást kapunk, amely hatékonyabban tud vezérlő prioritásokat létrehozni. Az így generált prioritásokkal a felépítő séma és a kiértékelést végző visszacsatolás során hatékonyabban találtam végrehajtható ütemterveket (6.2.2 alfejezet).

A kereső algoritmusom újdonságai a következők:

1. Egy új keresési stratégia a keresési tér bejárására.
2. Egy módosított eljárás új egyedek (megoldás jelöltek) létrehozására.
3. Egyedi prioritású, egyedi optimalizálási irányú célfüggvények együttes alkalmazása.

A kifejlesztett új algoritmust MOSM (*Multi-Objective Scheduling Method*) néven publikáltam [S18]. Az MOSM algoritmus pszeudo kódját mutatja be a 4. algoritmus. A jelölésrendszerben az $M_{i,g}$ jelöli az i-edik egyed feladatválasztás vezérlési értékeinek (CCV) vektorát a g-edik generációban. A kereső eljárás iteratívan határozza meg a $M_{i,g}$ értékeket és szimulációs eljárásként a generálási séma felépítő algoritmust használja a megvalósítható $SC(M_{i,g})$ ütemterv létrehozására.

A kereső algoritmusnak csupán két bemeneti paramétere van:

NM $NM \in \mathbb{Z}^+$, egy keresési iterációban generált populáció mérete.

NG $NG \in \mathbb{Z}^+$, az alkalmazott generációk száma.

	Algoritmus: MOSM
1:	{
2:	Bemeneti adatok betöltése, változók inicializálása;
3:	i = 1; g = 1;
4:	Amíg (i <= NM)
5:	{
6:	Számold ki az $M_{i,g}$ vektor értékeit a párhuzamos generálási séma és a legkorábbi indítható feladat szabályával;
7:	Hozd létre az $SC(M_{i,g})$ ütemtervet;
8:	Értékeld ki az $SC(M_{i,g})$ ütemtervet;
9:	Add az $M_{i,g}$ vektort az 1. generáció i -edik egyedének;
10:	i = i + 1;
11:	}
12:	Válaszd ki az 1. generáció legjobb egyedét;
13:	g = g + 1;

14: Amíg (g <= NG)
15: {
16: i = 1;
17: limit = 1 – g / NG;
18: Amíg (i <= NM)
19: {
20: random = generálj pszeudo véletlen számot a [0,1] intervallumból egyenletes eloszlás szerint;
21: Ha (random < limit)
22: Hozz létre egy új $M_{i,g}$ vektort az előző generáció i. elemének mutációjával;
23: egyébként
24: Hozz létre egy új $M_{i,g}$ vektort az előző generáció legjobb egyedének mutációjával;
25: Hozd létre az $SC(M_{i,g})$ ütemtervet;
26: Értékeld ki az $SC(M_{i,g})$ ütemtervet;
27: Hasonlítsd össze az $M_{i,g}$ és az $M_{i,g-1}$ kiértékelt ütemtervet és a jobbat add hozzá i-edik egyedként a g-edik generációhoz;
28: i = i + 1;
29: }
30: Válaszd ki a legjobb elemet a g-edik generációból;
31: g = g + 1;
32: }
33: Térj vissza az utolsó generáció legjobb egyedével;
34: }

4. algoritmus - Multi-Objective Scheduling Method

Az MOSM kereső eljárás generáció alapú evolúciós stratégiát alkalmaz, azonban nem keresztezési operátorokat, hanem csupán egy mutációs operátort használ. A keresési változók $M_{i,g}$ reprezentációját ebben a megközelítésben az ütemezési feladatban szereplő feladatok (taszkok) fontossági sorrendje jelenti. A g -edik generációban NM számú egyed kerül generálásra, mint lehetséges megoldás. A mutációs operátor két véletlenszerűen választott vektor elemet (taszkot) megcserél a kiválasztott $M_{i,g}$ vektorban. Fontos megjegyezni, hogy ezek a véletlen cserék azonban olyan feladatprioritási sorrendeket is eredményezhetnek, amelyek nem biztos, hogy megfelelnek a feladatok előfeltételi sorrendjének. A felépítő algoritmus ezeket tudja kezelni és mindig csak végrehajtható ütemtervet állít elő. Azonban a belső korrekcióról a kereső nem értesül, így a folytatásban az eredeti sorrendet használja és fejleszti tovább. A keresési hatékonyság növelése érdekében az ilyen esetek kifinomultabb kezelésére bevezetek egy precedencia sorrend normalizációs eljárást (*Normalization Algorithm – NA*).

4.5.3.1 Precedencia sorrend normalizációs eljárás (NA)

A precedencia sorrend normalizációs eljárás a feladatok előfeltételeit figyelembe véve minden egyes $M_{i,g}$ feladatsorrend-vektort megvizsgál és szükség esetén úgy módosít, hogy az aktuális feladat sorrendet a lehető legnagyobb mértékben megőrizze és a normalizációs eljárás végén minden egyes feladat vektorbeli pozíciója előtt szerepeljen az összes előfeltételi feladata. Ennek a normalizációs eljárásnak két célja van. Az első cél az, hogy a precedencia korlátok betartásából adódó feladatsorrend módosításról információt kap a kereső és ezt megőrizve a továbbhaladásban előnyösen fel tudja használni. A másik cél az, hogy ha már a keresőben megtörténik ez a normalizálás, akkor a vezérelt felépítő algoritmus felesleges vizsgálatokat ne végezzen, vagyis gyors direkt generálási sémavariánst használjon.

Az NA eljárás egyik bemeneti paramétere az $M_{i,g}$ prioritási vektor, amely a feladatok azonosítóját tartalmazza fontossági sorrendben. Egy másik bemeneti információforrás a feladatok előfeltételi korlátait biztosítja.

A normalizálási eljárás a bemenetként kapott vektort index szerint növekvő sorrendben kezdi el bejárni. Az aktuális sorszámú pozícióban lévő feladat esetén megvizsgálja, hogy a feladat ellenőrzött-e. Ha igen, akkor átlépi. Ha még nem ellenőrzött akkor megvizsgálja, hogy a feladatnak van-e még nem ellenőrzött előfeltételi (*predecessor*) feladata. Ha nincs, akkor a feladat ellenőrzötté válik és az azt követő (*successor*) feladatok esetén a még aktív előfeltételek listája frissítésre kerül, majd ezután az algoritmus lép a következő pozícióra. Amennyiben egy feladatnak még van nem ellenőrzött előfeltételi feladata, akkor az aktuális vektorbeli feladatsorrendet figyelembe véve megvizsgálja, hogy melyiknek a legkisebb a vektorbeli sorszáma (legnagyobb az ütemezési prioritása). Ha ez a feladat a vektorban kisebb sorszámú pozícióban található, akkor az NA algoritmus a korábbi pozíciótól kezdve folytatja az ellenőrzést. Egyébként lép tovább. Az ellenőrzés akkor ér véget, amikor a vektor utolsó pozíciójában lévő feladat is ellenőrzötté válik. A normalizált feladatsorrendet a feladatok ellenőrzötté válásának sorrendje adja meg.

4.5.3.2 Kiterjesztett felépítő eljárás a feladatok prioritási sorrendje alapján megvalósítható ütemezések generálására

Az MOSM eljárás előre meghatározott felépítő algoritmusokat használ a végrehajtható ütemterv generálására. Az ütemterv építő algoritmus (*Schedule Constructing - SC*) egy üres ütemtervből indul. Minden iterációban egy új, megfelelő rész-ütemterv kerül konstruálásra a választott feladat beillesztésével a korábbi rész-ütemtervbe. Ez a folyamat addig ismétlődik, amíg az összes feladat ütemezve lesz. Az eljárás pszeudokódját ismerteti az 5. algoritmus.

Algoritmus: Ütemtervépítő módszer

```
1:  {
2:  Bemeneti adatok betöltése;
3:  Üres ütemterv létrehozása;
4:  j = 1;
5:  Amíg ( j <= ütemezendő feladatok száma )
6:  {
7:          t_c feladat választása a j. pozíción az M_{i,g} vektorból;
8:          A t_c feladat beszúrása az ütemtervbe a legkorábbi indítási időponttal;
9:          j = j + 1;
10: }
11: Térj vissza az ütemtervvel;
```

5. algoritmus - Ütemterv Létrehozó Eljárás (SC)

Az SC egy speciális generálási sémának tekinthető, amely segítségével kezelhetjük a felépítés első (döntési halmaz képzési) és második (ütemezendő feladat kiválasztási) döntéshozatali problémát. Az $M_{i,g}$ feladatsorrend meghatározása a keresési algoritmus által történik, és a feladatok a megadott sorrendben kerülnek ütemezésre. A korábban ismertetett NA normalizáló algoritmus biztosítja, hogy a megelőző feltételek teljes mértékben teljesüljenek, így azt a felépítő eljárás során már nem kell ellenőrizni, vagy a legkorábbi feladat indítási időpontot annak megfelelően módosítani. Az SC algoritmusban tehát nem kell a döntési halmazt generálni és abból beütemezendő feladatot választani iteratívan, mert a bemeneti feladatsorrendet kell csupán követni. Az SC módszer kiszámolja, hogy a soron következő feladat mikor kezdődhet el legkorábban és aszerint ütemezi a feladatot. Ebben a konkrét kezdési időt számító eljárásban az SC ugyanazt a logikát követi, mint az FCMPGS.

Az SC ütemterv felépítő algoritmus funkcionális szempontból az FCMPGS algoritmussal helyettesíthető. Ehhez az FCMPGS bemeneti GSTP vektorát úgy kell konfigurálni, hogy csupán a T_{cPrio} feladatválasztási elvet vegye figyelembe és az optimalizálási irány a minimalizálás legyen (a kisebb érték jelzi a fontosabb feladatot). Ekkor az FCMPGS futási eredménye csak a kereső modul által generált CCV értékektől függ. A CCV értéket pedig úgy kell megadni, hogy a MOSM módszer által előállított $M_{i,g}$ feladatsorrendből kiindulva minden egyes feladat a sorrend szerinti sorszámot kapja meg prioritási értékként.

Az SC és az FCMPGS módszerek ugyanazt az ütemtervet állítják elő. Az SC előnye, hogy sokkal gyorsabb működést biztosít azáltal, hogy a megelőzési relációk ellenőrzését nem végzi el, mert a precedencia korlátozásokat az NA algoritmus biztosítja.

A bemutatott MOSM módszert az NA és az SC algoritmusokkal együtt alkalmazva teszteltem különböző feladatokon és összehasonlítottam más ismert algoritmusokkal is. A tesztekre mutat példákat az értekezés 6. fejezete. Az MOSM, NA és SC algoritmusokkal kapcsolatos eredményt a 4. tézis foglalja össze az értekezés 7. fejezetében.

5 A megoldó módszer szoftveres megvalósítása

Az ismertetett koncepció hatékonyságának ellenőrzésére két különböző szoftveres implementáció készült. Az első megoldás C++ nyelven (C++98) készült, és főként a hibrid megoldó algoritmusok teljesítményének vizsgálatára összpontosított. A C++ nyelv rugalmas és hatékony programozási nyelv, amely ideális választás lehet a különböző algoritmusok összehasonlítására és a hatékonyság optimalizálására. A C++ megoldás célja az algoritmusok gyors implementációja volt, és ezt a szoftvert alkalmaztam a tesztek és a mérések elvégzésére.

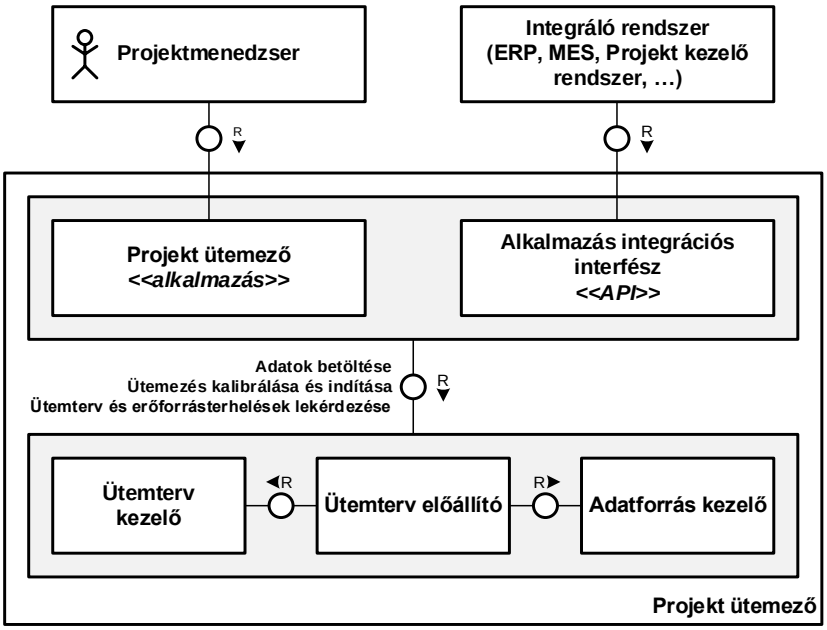
A második megoldás ABAP nyelven készült. Az ABAP programozási nyelv az SAP vállalatirányítási rendszer elterjedt verzióinak saját programozási nyelve. Támogatja a procedurális és az objektum-orientált szemléletű programozási paradigmákat. Az ABAP nyelven készült implementáció célja a dinamikusan bővíthető célfüggvények, a rugalmas konfigurálhatóság és a többcélúság együttes alkalmazásának megvalósíthatósági tanulmányozása volt. Az ABAP nyelven készített verzió nem csupán egy szoftverimplementációt jelentett, sokkal inkább egy szoftver-architektúrális koncepció kidolgozásának tekinthető. Az ABAP nyelven készített verzió figyelembe veszi a korszerű szoftvertechnológiai szempontokat, többek között az egyes komponensek hatékony függetlenítését és cserélhetőségét, a modulok és alkomponensek önálló tesztelhetőségét, valamint az elterjedt vállalatirányítási rendszerek integrálhatóságát is. A következő alfejezetekben ennek a szoftveres megoldásnak a fő architektúrális tervezési elveit mutatom be.

5.1 Koncepcionális felépítés

A szoftvertervezés során objektum-orientált megközelítést alkalmaztam. Az implementált kód nem tartalmaz olyan ABAP specifikus programnyelvi elemet, amelyet ne lehetne könnyen leképezni más objektum-orientált programozási nyelvre. Azért választottam ezt a megközelítést, hogy az elkészített szoftver-architektúra könnyen átültethető legyen más objektum-orientált programozási nyelvekre, és általános maradjon.

A modern SAP rendszerekben az SAP memória alapú adatbázisára (SAP HANA) optimalizált megoldásokat alkalmaznak. Ennek egyik fő oka, hogy hatékony adatfeldolgozás történik az adatbázisban, főként deklaratív programozási elemek által, csökkentve az applikációs szerverek terhelését, és így összességében jobb rendszerteljesítmény érve el. A korábban említett okok miatt ezeket a fejlesztési eszközöket az implementációmban nem alkalmaztam, így ipari alkalmazás esetén érdemes megfontolni az implementált megoldás kibővítését és adaptálását.

A RCMOMPSP probléma megoldó szoftveres megvalósításának magasszintű felépítését mutatja be a 7. ábra.



7. ábra - Az ABAP implementáció koncepcionális felépítése

A projektütemezőt két hozzáférési csatornán keresztül lehet elérni:

- Projekt ütemező alkalmazás (*Application – App, User interface – UI*)
  A projektütemező alkalmazás egy modellezett projektmenedzser szerepkörrel rendelkező felhasználó számára érhető el. Az alkalmazás rendelkezik grafikus felülettel, ahol a felhasználó létrehozhatja az ütemezési problémát, vagy azt integrált adatforrásból beolvashatja azt. A probléma definiálása mellett az ütemezés céljait és prioritásait lehet meghatározni. Lehetőség van a projektütemező kalibrálására, az ütemezési eljárás finomhangolására. Az ütemezés elindítása után a felület alkalmas az elkészített ütemterv, az erőforrások tervezett terhelésének és a célfüggvények értékeinek a megjelenítésére.

- Alkalmazás programozási interfész (*Application Programming Interface - API*)
  Rendszeren belüli vagy rendszeren kívüli integráció szempontjából egy dedikált alkalmazás programozási interfészt terveztem. Ennek szerepe, hogy előkészítse a meglévő projektkezelő rendszerek vagy más ipari alkalmazások integrációját. A dedikált technikai interfész lehetővé teszi az ütemező szoftver életciklusának elkülönített kezelését, valamint különböző kompatibilitási szerződések (*stability contract*) kialakítását az integráció során. Egy rendszer-rendszer integráció esetén egy inkompatibilis bővítés vagy javítás nem okozhatja a meglévő integrációk „törését”, így interfészek verziózására és a kompatibilitás garantálására van szükség egy előre meghatározott garanciális időszakra.

Mind a felhasználói felület, mind az API(k) egy közös projektütemező modult használnak. A projektütemező három fő almodulra tagolható: (1) adatforráskezelő almodul, (2) ütemterv előállító almodul, és (3) ütemtervkezelő almodul.

A kialakított almodulok szerepe az, hogy az ütemezési probléma modellezését, az ütemtervet előállító algoritmust és az ütemezés eredményeként előállított ütemtervet külön felelősségi körrel rendelkező komponensek valósítsák meg. Az almodulok egymással lazán integráltak, a kommunikáció modellezett interfészeken keresztül történik. Ezáltal az ütemtervet előállító almodul a probléma adatainak eléréséhez az adatforráskezelőt, mint csak olvasási operációkat támogató interfészt, az ütemterv kezelőt, mint csak írási operációkat támogató interfész éri el, és közvetlen adattároló hozzáféréssel nem rendelkezik.

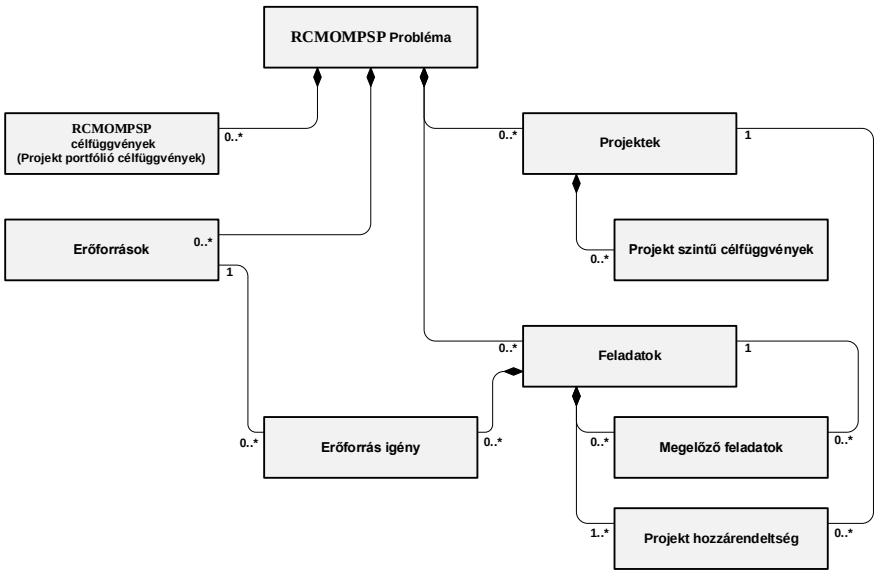
Az implementációm során a belső ágensek egyetlen szolgáltatásként működnek, azonban korszerű microservice alapú környezetben az almodulok független szolgáltatásonként is megvalósíthatóak. Ennek előnye, hogy az almodulok külön-külön megfelelő technológiai környezettel működhetnek, valamint az életciklusokat függetlenül lehet kezelni. Fontos azonban biztosítani a microservice-k közötti hatékony adatkommunikációt és verziómenedzsmentet.

5.2 Adatforrás kezelő

Az adatforráskezelő szerepe, hogy a 3. fejezetben bemutatott modellt egy belső adatszerkezetben elérhetővé tegye. Ehhez a matematikai modellt egy entitás-relációs modellre képeztem le, majd kialakítottam egy adatmodell kezelő ágenst.

5.2.1 Az RCMOMPSP modell leképzése E/R modellre

Az RCMOMPSP matematikai formalizmusát egy entitás-relációs modellre képeztem le, amelyet a 8. ábra szemléltet. A bemutatott entitás-relációs modell alkalmas arra, hogy a RCMOMPSP feladatokat tárolja akár futásidőben a memóriában, vagy perszisztens módon az adatbázisban.



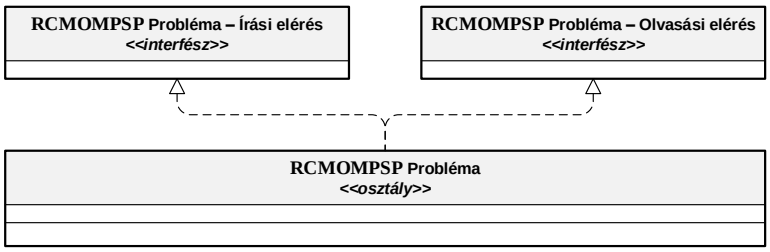
8. ábra - Az RCMOMPSP modell entitás-relációs modellje

Az implementáció során az ABAP adatszótárt alkalmaztam. Más programozási eszközökben ez a struktúrák, típusok, vagy osztályok használatával érhető el.

5.2.2 Adatelérési objektum-orientált modell

Az erősen tagolt objektum-orientált megközelítésben minden entitás egy objektumnak tekinthető, ami az adott entitás egy példányára vonatkoztatva kezeli az állapotteret és definiálja a végrehajtható műveleteket. Ennek előnye a finom modellezettség, azonban hátránya lehet a nagyobb futásidejű memória igény és az üzenetküldési (eljárás felhívási) költség. Kevésbé tagolt esetben az entitások összevonhatóak, amikor az állapottér közös kezelését egy osztály valósítja meg.

Megközelítésemben a probléma leírásával kapcsolatos műveleteket és állapottér kezelést a modell egészén értelmeztem, ezért a teljes RCMOMPSP modellt egy saját osztályban zártam egységbe. A modellen belül az ismertetett E/R modell egy összetett adatstruktúrával kerül megvalósításra. A koncepcionális áttekintőben ismertetett almodulban az RCMOMPSP probléma kezelésére dedikált írási és olvasási interfészeket hoztam létre. Az adatelérési modul feladata, hogy az írási interfész használatával a forrás problémát a modellre képezze, és biztosítsa a modell szerinti adatkonzisztenciát. Az ütemező almodul csak a dedikált olvasási interfészen keresztül érheti el az ütemezési probléma adatait.



9. ábra - RCMOMPSP model egyszerűsített objektum-orientált megvalósítása

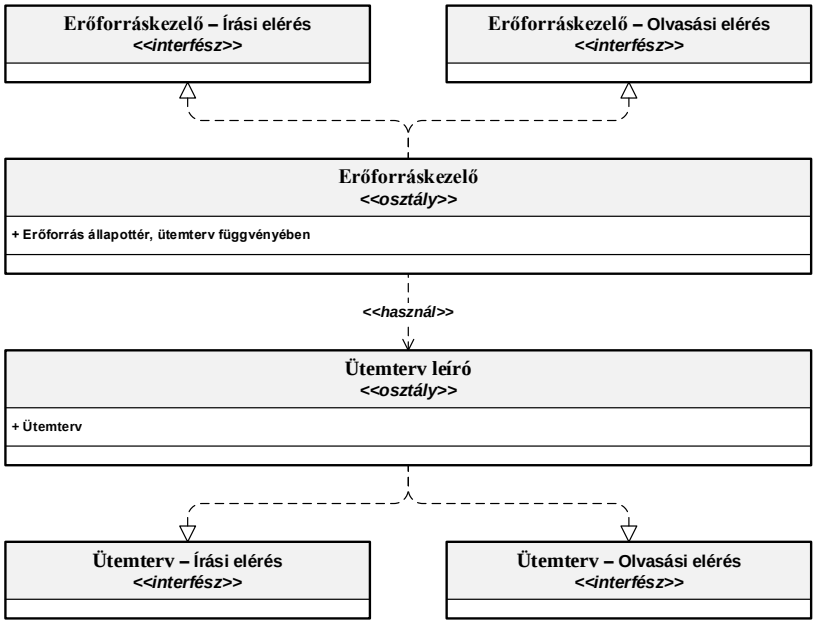
5.3 Ütemterv kezelő és az erőforrások reprezentálása

Egy RCMPMOSP probléma egy végrehajtható megoldását az ütemterv reprezentálja. A matematikai modellben bevezetett SCH ütemterv könnyen leképezhető egy egyszerű E/R modellre. Minden feladat esetén rögzítjük a feladat kezdési idejét, amely így egyértelműen tárolja az elsődleges döntési változók esetén kiszámított értékeket.

Az ütemterv könnyebb reprezentálhatósága miatt másodlagos számított értékek is eltárolásra kerülnek. Ez a redundáns adattárolás a jobb futásidő elérését szolgálja. Például redundánsan tárolt adat lehet egy feladat maximális végrehajtási ideje, a feladat befejezési ideje a CPM módszer által, a feladat aktuálisan számított lekorábbi befejezési ideje, a feladat megelőző és követő feladatai és így tovább.

Az erőforrások terhelése egy dedikált erőforráskezelő ágensben kerül megvalósításra. Az erőforráskezelő feladata az elérhető erőforrások kapacitáskorlátainak rögzítése, valamint az

ütemezés során kialakuló terhelések menedzselése. Az erőforrások terhelése ütemezésenként változhat, ezért az erőforráskezelő terheltségkezelése mindig az aktuális ütemterv függvénye.



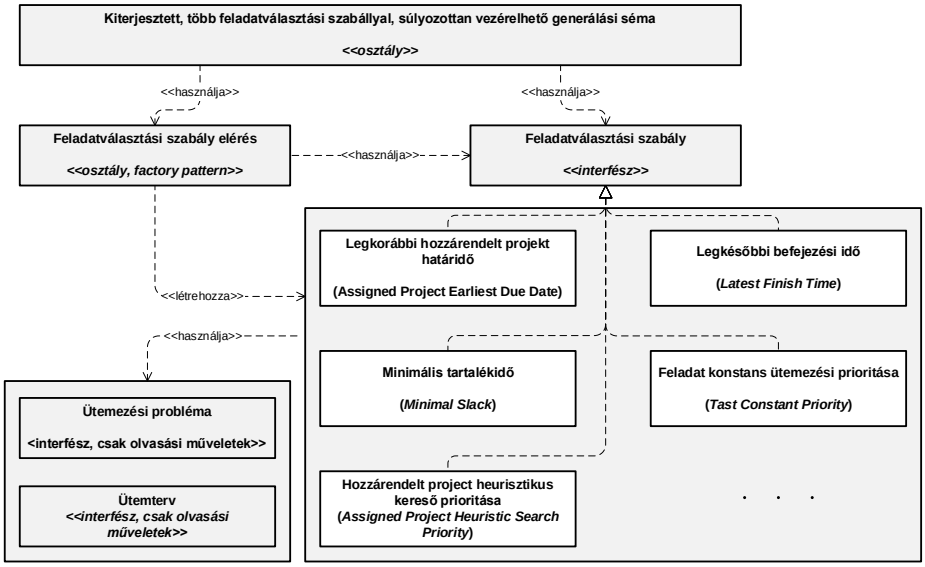
10. ábra - Erőforráskezelő és ütemtervleíró egyszerűsített objektum-orientált modellje

5.4 Ütemterv előállító modul

Az ütemezési probléma és az erőforrás menedzser és ütemterv között a kapcsolatot az ütemtervet előállító modul valósítja meg. Ez a modul több fő komponensből áll.

5.4.1 FCMPGS eljárás megvalósítása

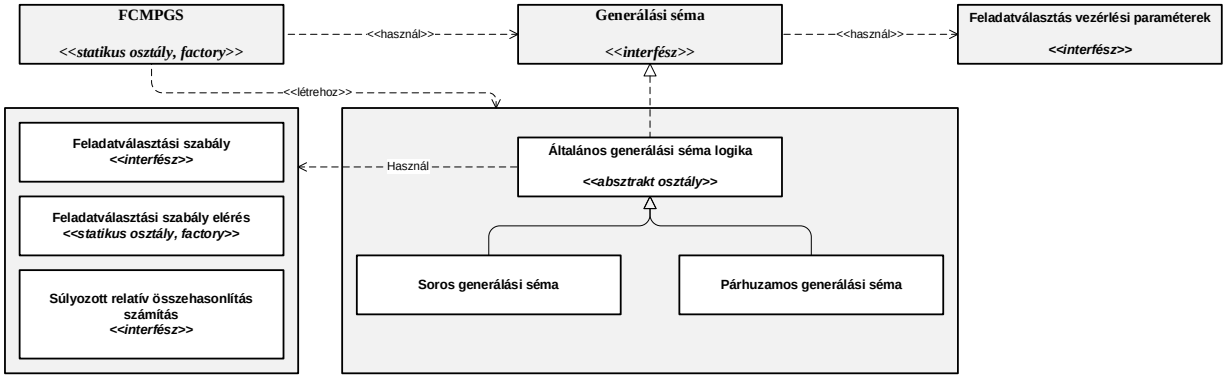
Az FCMPGS eljárás objektum-orientált modellezése és megvalósítása során az alábbi szempontok kerültek figyelembevételre: (1) a felépítő eljárást közvetlenül, a későbbiekben ismertetésre kerülő kereső modul nélkül is lehessen alkalmazni ütemtervek előállítására, (2) a generálási séma variáns (soros, párhuzamos) implementációja rejtett és közvetlenül nem használható, (3) a kiterjesztett, több feladatválasztási szabállyal, prioritásokkal vezérelhető generálási séma esetén a feladatválasztási szabályok halmaza nem korlátos, (4) az általános generálási séma algoritmusának feladatválasztási lépése több feladatjelölt esetén a súlyozott relatív minősítő *F* függvény szerint történik. A tervezett modul koncepció szintű osztály-diagrammját mutatja be a 11. ábra.



11. ábra - Feladatválasztási szabályok implementációs terve

Generálási séma variánsok

Az alfejezet bevezetőjében ismertetett (2) szempont alapján a generálási sémák implementációi közvetlenül nem elérhetőek, ezért terveztem egy elérési osztályt, amely egy gyártó mintával (*factory pattern*) készült a generálási séma variánsának kiválasztására. Ennek az osztálynak a feladata, hogy egy paraméter segítségével meghatározzuk a generálási séma variánsát, valamint kiválassza és példányosítsa a konkrét implementációt. Ennek a megközelítésnek köszönhetően lehetővé válik, hogy a GS-t alkalmazó modulok a GS-ek esetén az elvárt interfészek szerint kerüljenek megvalósításra, míg a séma specifikus részletek rejtve maradnak. Ezzel a megközelítéssel az alkalmazott variáns akár a heurisztikus kereső által is meghatározható döntési változó is lehet. Természetesen – alkalmazási területtől függően – ez lehet rögzített paraméter is. A soros és párhuzamos generálási séma algoritmusa a döntési halmaz előállításában különbözik, de az algoritmusuk jelentős része azonos. Ennek leképezésére a közös logikát egy absztrakt osztályban hoztam létre, valamint delegációs mintával (*delegation pattern*) tettem lehetővé a variáns specifikus algoritmusok megvalósítását.



12. ábra - Generálási sémák implementációs terve

A generálási séma lehetővé teszi annak megadását, hogy milyen módon válasszon a feladatok közül, ha több, mint egy végrehajtható feladat áll rendelkezésre az ütemterv generálásakor. Amennyiben nem kerül meghatározásra a feladatválasztási paraméter, a generálási séma determinisztikusan a feladatok egyedi kulcsa alapján választ.

5.4.2 Feladatválasztási szabályok

Minden feladatválasztási szabály esetén elvárt, hogy a szabály kiértékelése egy valós számot eredményezzen.

Statikus és dinamikus feladatválasztási szabályok

A 4.3.2.2 fejezetben bemutatott módon a feladatválasztási szabályokat két kategóriába sorolhatjuk: statikus szabályok és dinamikus szabályok. A feladatválasztási szabály mindkét esetben a generálási séma számára ugyanazon a dedikált interfészen keresztül érhető el. A kiterjesztett generálási séma ezen interfész metódusait hívhatja meg egy adott feladat esetén egy adott feladatválasztási szabály prioritás értékének kiszámítására.

Ahogy a 12. ábra is szemlélteti, a feladatválasztási szabályok nem integrált részei a kiterjesztett generálási sémának. Egy új feladatszabály bevezetése így nem teszi szükségessé a generálási séma kiterjesztését, hanem mindössze egy új feladatválasztási szabály algoritmusát kell megvalósítani, valamint a feladatválasztási szabályt gyártó osztálynak kell azt egy szabály azonosítóhoz rendelni.

Mind statikus, mind dinamikus feladatválasztási szabály esetén előfordulhat, hogy a szabály kiértékeléséhez az ütemezési probléma adataira, vagy az aktuálisan elkészített rész-ütemterv adataira is szükség van. A feladatválasztási szabály önmaga elérése egy hozzáférést biztosító (*factory*) osztály által megadott olvasási interfészeken keresztül van megoldva és így érhetőek el a probléma vagy a (rész)ütemterv olvasási interfészei is.

Súlyozott feladatválasztás a $\boldsymbol{D_m}$ döntési halmazból

Ahogy a 12. ábra bemutatja, egy általános generálási séma logikát implementáló absztrakt osztályt hoztam létre. Az egyes GS variánsok csupán a döntési halmaz létrehozásának logikájában különböznek, ezért variánsonként egy-egy gyerek osztályt hoztam létre. A gyerek osztályokban a döntési halmaz létrehozásának eljárását delegációs mintával lehet specializálni.

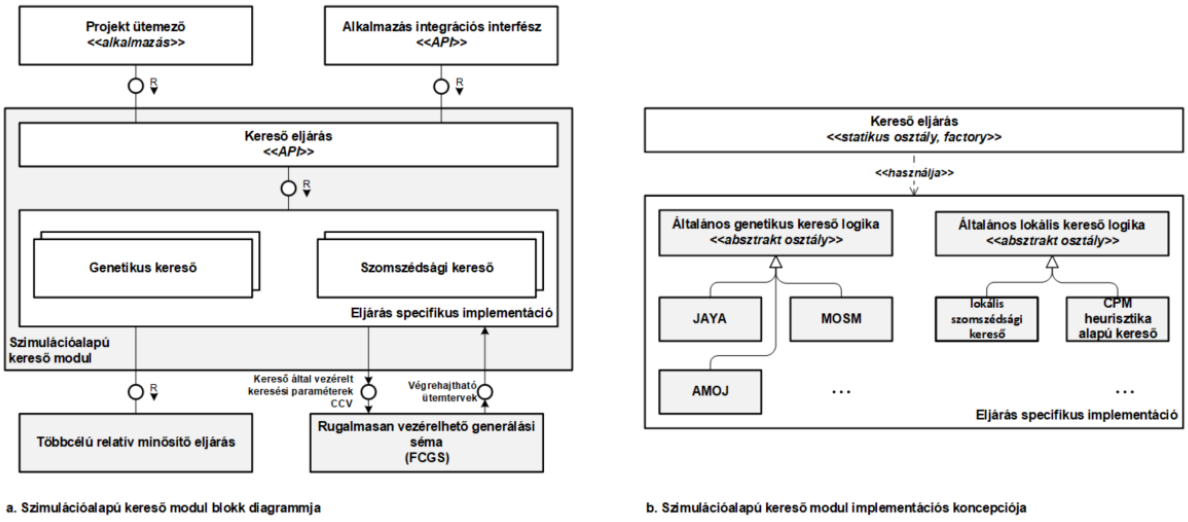
A döntési halmazból történő feladatválasztás lépésében (3. döntési fázis) az egyes variánsok nem különböznek. A súlyozott relatív feladatválasztást egy kiszervezett, állapotmentes F függvényt megvalósító „súlyozott relatív összehasonlítás számítás” osztály eljárása biztosítja. A kiszervezés oka, hogy ugyanezt az eljárást alkalmazza a heurisztikus kereső is. Az F függvény alkalmazásához szükséges, hogy minden alkalmazott feladatválasztási szabály esetén az adott szabály aktuális értékei kiszámításra kerüljenek. Ez a fentebb kifejtett interfészen keresztül tehető meg. A feladatválasztási szabályok prioritása és optimalizálási iránya az ütemezés során nem változik. A generálási séma meghívásakor megadott vezérlési paraméterek interfészen keresztül érhetőek el.

5.4.3 Kereső eljárás megvalósítása

A tervezett kereső almodul koncepciójának osztály-diagrammját mutatja be a 13. ábra. Hasonlóan az FCMPGS-hez a kereső implementációjában szempont volt, hogy a konkrét algoritmusok ne közvetlenül, hanem egy dedikált gyártó eljáráson keresztül lehessen elérni. Ennek oka, hogy egy adott implementáció később az integrációs komponensek számára transzparensen cserélhető legyen.

A kereső eljárásokat két fő csoportra osztottam: (1) genetikus kereső eljárások, (2) heurisztikus kereső eljárások. A genetikus keresők elvi felépítése nagy hasonlóságot mutat és a specialitás több esetben az új populáció létrehozásában van. A közös implementáció kezelésére ezért egy absztrakt osztályt hoztam létre, amelyben delegációs mintát használtam a konkrét implementáció specifikus elemek létrehozására. Például az AMOJ és a JAYA eljárások egymástól csak az új egyed létrehozásában különböznek.

Hasonlóan a heurisztikus keresőknek is több közös implementációja van, ezért a mintát követve ott is egy általános kereső ősosztályt alakítottam ki.



13. ábra - A kereső implementálási terve

Hasonlóan az FCMPGS-hez a lehetséges végrehajtható ütemtervek összehasonlítását az F függvénnyel végeztem. Ehhez az egyes célfüggvények értékét ki kell számítani, majd a többcélú relatív minősítő eljárást meghívni. Amennyiben a keresés során a kereső a CCV értékeket módosítja, úgy az FCMPGS eljárás, mint konstruktív szimulációs eljárás többször meghívható.

6 Futási eredmények és a módszerek validálása

A kidolgozott megoldó módszer hatékonyságának vizsgálatára több megközelítést alkalmaztam. Az első esetben olyan ismert gyártásütemezési problémákat képeztem le az RCMOMPSP modellre, amelyek esetében ismert egy polinomiális időben végrehajtható, a problémában definiált célfüggvény szerinti optimumot adó ütemezési algoritmus. A második esetben NP nehéz projektütemezési benchmark problémákat képeztem le az RCMOMPSP modellre. A benchmark feladatok elérhetőek olyan publikált források, amelyek tartalmazzák az adott feladat példány esetén közzétett bizonyított alsó, vagy felső korlátokat, ha elérhető a bizonyított optimum érték, illetve ennek hiányában az eddig ismert legjobb megoldás. Végül az RCPSP benchmark problémákból kiindulva többprojektes, több célfüggvényt egyszerre figyelembe vevő problémákat állítottam elő, amelyeken teszteltem az ütemező hatékony vezérelhetőségét a célfüggvények prioritásaival és mértem az egyes célfüggvények tekintetében elért eredményeket. Ellenőriztem, hogy a több feladatválasztási szabályt alkalmazva az ütemterv felépítő generálási séma az elvárt teljesítményt és viselkedést mutatja-e.

A hibrid keresők teljesítményét minden esetben tesztsorozatok alkalmazásával vizsgáltam, mivel a kereső modul jellegéből adódóan véletlenszerű műveleteket (pl. kiindulási megoldások generálását, szomszédok készítését, mutációs műveleteket, stb.) is figyelembe kellett vennem. A kereső teljesítményértékelése alatt a teszt során a megtalált ütemtervek jóságát tekintettem és nem az ütemterv előállításának számítási kapacitását vagy a számítás elvégzéséhez szükséges futásidőt.

6.1 Validálás polinomiális időben megoldható ütemezési feladatokkal

Az első megközelítésben ismert gyártásütemezési feladatokat vizsgáltam. A feladatok jelölése az $\alpha|\beta|\gamma$ jelölésrendszert használtam, ahol α a feladat jellegét, β a megkötéseket és γ a célfüggvényt jelöli. A szakirodalomban ez a jelölésrendszer az egyik legelterjedtebb formalizmus [68].

6.1.1 Futási eredmények generált $F2||C_{max}$ feladatok esetén

A két gépes, előzésmentes, a legutolsó feladat befejezési idejét minimalizáló flowshop ütemezési problémát formálisan $F2||C_{max}$ módon adhatjuk meg. A gyártásütemezési feladat leírásában használt kiindulási adatok jelölése:

$\boldsymbol{J}$ $\boldsymbol{J} = (j_1, j_2, \dots, j_i \dots, j_n)$, *n* számú ütemezendő munka.

$\boldsymbol{O}$ $\boldsymbol{O} = (o_{11}, o_{12}, o_{21}, o_{22}, \dots o_{i1}, o_{i2}, \dots, o_{n1}, o_{n2})$ operációk. Minden munka két operációból áll. Az o_{i1} operáció végrehajtása megelőzi az o_{i2} operáció végrehajtását.

m_1, m_2 Két dedikált gép áll rendelkezésre az operációk végrehajtására. Minden munka első operációját $(o_{i1}, 1 \leq i \leq n)$ az m_1 gépen kell végrehajtani, míg minden munka második operációját $(o_{i2}, 1 \leq i \leq n)$ az m_2 gépen kell végrehajtani. Minden gép

egyszerre csak egy operációt tud végrehajtani. Egy munkán egyszerre csak egy operáció hajtható végre.

p_{i1}, p_{i2} Az operációk végrehajtásának idejét jelöli rendre az o_{i1} o_{i2} operációk esetén az i munka esetén.

C_{max} A célfüggvény az utolsó feladat befejezési idejének (*makespan*) minimalizálása.

Az $F2||C_{max}$ problémára Selmer Martin Johnson publikált egy optimális megoldást adó ütemezési algoritmust [69].

Állítás: az $F2||C_{max}$ütemezési probléma az RCMOMPSP modellre leképezhető.

Bizonyítás: Létezik olyan módszer, amely az $F2||C_{max}$ ütemezési problémát egyértelműen a RCMOMPSP problémára leképezi. A leképzés az alábbiak szerint végezhető el:

- Minden munka egy projektnek feleltethető meg. $\boldsymbol{P} := \boldsymbol{J}$
- Minden operáció egy egyedi feladatra képezhető le.
  Az így létrehozott feladat ahhoz és csak ahhoz a projekthez tartozik, amely projektet ahhoz a munkához hoztunk létre, amelyhez a feladat kiindulásaként tekintett operáció tartozott.
$$\boldsymbol{T} := \boldsymbol{O}, \boldsymbol{T} = \{t_1, t_2, \dots, t_{2n}\} \mid t_1 = o_{11}, t_2 = o_{12}, \dots, t_{2n-1} = o_{n1}, t_{2n} = o_{n2}$$
$$\boldsymbol{TA} = \{(p_i, t_{2*i}), (p_i, t_{2*i+1})\} | 1 \leq i \leq n$$
- A projekt első feladata előfeltétele a projekt második feladatának.
$$\boldsymbol{PRE}_j = \begin{cases} \emptyset, ha\ j\ páratlan, \\ \{t_{j-1}\}, ha\ j\ páros \end{cases}$$
- Minden gép egy saját erőforrástípusra képezhető le, amely erőforrástípusnak konstans egy egység kapacitása van. $\boldsymbol{R} = \{r_1, r_2\},\ RC_{r_1} = 1,\ RC_{r_2} = 1$
- Az első operáció csak az első gépet, míg a második operáció csak a második gépet foglalja. Ezáltal a leképzés során minden operáció megfeleltethető egy nem virtuális feladatnak, amelynek rendre az első, vagy a második erőforrástípus szerint 1 egységnyi erőforrás igénye van, az eredeti operációban megadott feldolgozási időig.
$$rr_{2*i,1} = 1;\ rr_{2*i+1,2} = 1;\ pt_{2*i,1} = p_{i1};\ \ pt_{2*i+1,2} = p_{i2};$$
$$(1 \leq i \leq n).$$
- Az RCMOMPSP modellben csak egyetlen célfüggvény kerül meghatározásra, tetszőleges nem nulla fontossági súllyal. A célfüggvény a projektek befejezési idejének minimalizálása: $f_{PC_{max}}$

Tesztelés: A tesztelés során 100 darab $F2||C_{max}$ ütemezési feladat került generálásra. Minden ütemezési feladat 50 munkából, azaz 100 operációból állt. Az egyes operációk végrehajtási ideje egy egyenletes eloszlás szerint véletlenszerűen választott egész szám a [1,100] intervallumból. Minden ütemezési feladat esetén az optimum érték kiszámításra került a Johnson algoritmus alkalmazásával [69]. A leképzett RCMOMPSP problémára a hibrid

megoldási megközelítést (AMOJ + FCMPGS) alkalmaztam. A mérés során három tesztelési ciklus került végrehajtásra. A megoldás során alkalmazott keresési paramétereket mutatja be az 1. táblázat.

1. táblázat - $F2||C_{max}$ tesztelési beállításai

Paraméter	Érték 1. teszt ciklus	Érték 2. teszt ciklus	Érték 3. teszt ciklus
Keresési iterációk száma	500	500	1000
Populáció mérete	100	200	500
Ismételt futtatások száma	5	5	5

Egy tesztelési ciklus 5 teljes teszt végrehajtást tartalmaz. Minden végrehajtás során az AMOJ + FCMPGS hibrid eljárást mind a 100 ütemezési feladat példány esetén végrehajtottam. Egy teszt során a kereső minden esetben új, véletlenszerűen generált kezdeti populációt generált. Egy teszt ciklus összesített végeredményének az 5 teszt futási eredményei közül a célfüggvény szerinti legjobb futási eredményt tekintettem. A futási eredmények összevetését a Johnson algoritmus garantáltan optimumot adó megoldásával mutatja a 2. táblázat.

2. táblázat - $F2||C_{max}$ futási eredményeinek áttekintő összegzése

Kiértékelési nézőpont	Érték
Probléma példányok száma	100
Megtalált optimális ütemtervek száma	100
Találati hatékonyság	1.0
Átlagos relatív távolság az optimum értéktől	0
Maximális relatív távolság az optimum értéktől	0

Ahogy a 2. táblázatban látható a javasolt AMOJ és FCMPGS hibrid megoldó módszer minden ütemezési feladat esetén, minden tesztelési ciklusban legalább egyszer megtalálta az optimum megoldást. A teszt ciklus során használt keresési paraméterektől függően az optimum érték megtalálási gyakorisága eltért. A találati hatékonyságot mutatja be a 3. táblázat, amelyben minden tesztelési ciklusban az 5 végrehajtott teszt alapján megszámlálásra kerültek az ismert optimumot megtaláló esetek.

3. táblázat - $F2||C_{max}$ tesztelés eredményeinek optimum találati gyakorisága

Optimumot megtaláló tesztek száma	1. teszt ciklus	2. teszt ciklus	3. teszt ciklus
0	0	0	0
1	4	0	0
2	0	4	0
3	9	3	1
4	22	15	3
5	65	78	96

A 3. táblázat alapján megállapítható, hogy a teszt ciklusok túlnyomó többségében mind az öt teszt optimum értéket szolgáltatott. A teszt ciklusok egymáshoz viszonyított javuló eredményei a futásidőt növelő populáció, vagy generációk számának emelésével indokolható.

A harmadik tesztelési ciklusban a keresés iterációnak számát 1000-re, az alkalmazott populációs méretet 500-ra növeltem. Ahogy látható ez az optimumot megtaláló tesztek gyakoriságában jobb eredményeket jelentett.

Figyelembe véve, hogy a megoldó eljárás a probléma jellegéről semmilyen további információt nem vesz figyelembe (például nem használ kritikus utat, nem tekinti az erőforrások aktuális terhelését és nem él semmilyen egyéb AMOJ-ban nem közölt heurisztikával), úgy az elért eredményeket kiemelkedően jónak ítélem meg. Összevetésként a problémáról hasonlóan semmilyen információt nem használó „*brute-force*" jellegű stratégia esetén a megvizsgálandó esetek száma 50! lenne.

6.1.2 Futási eredmények generált $1||C_{sum}$ *feladatok esetén*

Az egygépes, a végrehajtandó feladatok összesített befejezési idejét minimalizáló ütemezési problémát formálisan $1||C_{sum}$ módon adhatjuk meg. Az ütemezési feladat megadásában használt jelölések az alábbiak:

$\boldsymbol{J}$ — $\boldsymbol{J} = (j_1, j_2, \dots, j_i, \dots, j_n)$, n számú ütemezendő munka. A munkák egymástól függetlenül végrehajthatóak. Minden munkának egyetlen operációja van.

m — Egy gép áll rendelkezésre az operációk végrehajtására. A gép egyszerre csak egy operációt tud végrehajtani.

p_i — Az *i.* munka végrehajtásának idejét jelöli az *m* gépen.

C_{sum} — Az ütemezés célfüggvénye a feladatok összesített befejezési idejének a minimalizálása.

$$C_{sum}(SCH) = \sum_{i \in J} C_i \tag{39}$$

Állítás: Az $1||C_{sum}$ ütemezési probléma az RCMOMPSP modellre leképezhető.

Bizonyítás: Létezik olyan módszer, amely az $1||C_{sum}$ ütemezési problémát egyértelműen a RCMOMPSP problémára leképezi. A leképzés az alábbiak szerint végezhető el:

- Minden munka egy egyedi projektnek feleltethető meg. $\boldsymbol{P} := \boldsymbol{J}$
- Minden operáció egy egyedi feladatra képezhető le. A feladat ahhoz és csak ahhoz a projekthez tartozik, amely projektet ahhoz a munkához hoztunk létre, amelyhez a feladat kiindulásaként tekintett operáció tartozott.

$$\boldsymbol{T} := \boldsymbol{O}, \boldsymbol{TA} = \{(p_i, t_i)\} | 1 \leq i \leq n$$

- A feladatok műveleti ideje megfelel az eredeti feladat operációinak műveleti idejével.
- A feladatok előfeltétel halmaza üres. $\boldsymbol{PRE}_j = \emptyset, \forall j \in \boldsymbol{T}$
- Egyetlen gép áll rendelkezésre, amelyet egyetlen erőforrás típusra lehet leképezni. Az így definiált erőforrástípusnak konstans egy egység a kapacitása.

$$\boldsymbol{R} = \{r_1\}, \qquad RC_{r_1} = 1$$

- Minden feladat az r_1 erőforrástípust használja 1 kapacitással.

$$rr_{j,1} = 1, \forall j \in \boldsymbol{T}$$

- Az RCMOMPSP modellben csak egyetlen célfüggvény kerül meghatározásra, tetszőleges nem nulla fontossági súllyal. A célfüggvény a projektek feladatainak összesített befejezési idejének minimalizálása. Ehhez egy új célfüggvény kerül felvételre az RCMOMPSP modellbe az alábbiak szerint:

$$f_{PCsum} = \sum_{j \in \boldsymbol{T}} C_j \quad (40)$$

Tesztelés: A tesztelés során 100 darab $1||C_{sum}$ ütemezési feladat került generálásra. Minden ütemezési feladat 30 munkából áll. Az egyes munkák végrehajtási ideje egy egyenletes eloszlás szerint véletlenszerűen választott egész szám a [1,100] intervallumból. Minden ütemezési feladat esetén az optimum érték kiszámításra került a „legrövidebb feldolgozási idejűt először" (*Shortes Processing Time – SPT*) algoritmus alkalmazásával [68].

A leképzett RCMOMPSP problémára két, hibrid megoldási megközelítést alkalmazó megoldóval generáltam megoldásokat. Mindkét esetben az FCMPGS algoritmust alkalmaztam konstruktív szimulációs komponensként. A kereső eljárásban genetikus algoritmust használtam. Az első teszt esetén a Rao által publikált JAYA eljárás módosító operátorát használtam a genetikus algoritmusban, míg a második esetben az AMOJ algoritmus módosító operátorát. Mindkét esetben ugyanazokat a keresési paramétereket alkalmaztam, amelyet a 4. táblázatban ismertetek.

4. táblázat - $1||C_{sum}$ ütemezési feladat tesztelési beállításai

Paraméter	Érték
Keresési iterációk száma	250000
Populáció mérete	40
Ismételt futtatások száma	5

A tesztelés során mind a 100 generált probléma megoldásra került. Mindkét hibrid kereső eljárás pontosan ötször került futtatásra minden ütemezési probléma esetén. A futási eredmények összevetését az SPT garantáltan optimumot adó megoldásával mutatja az 5. táblázat.

5. táblázat - $1||C_{sum}$ futási eredményeinek áttekintő összegzése

Kiértékelési nézőpont	Érték JAYA	Érték AMOJ
Probléma példányok száma	100	100
Megtalált optimális ütemtervek száma	100	100
Találati hatékonyság	1.0	1.0
Átlagos relatív távolság az optimum értéktől	0	0
Maximális relatív távolság az optimum értéktől	0	0

A hibrid megoldó eljárás mindkét módosító operátor alkalmazásával a teszt során minden ütemezési probléma esetén legalább egyszer megtalálta az optimum megoldást. A teszt során egy ütemezési probléma esetén 5 keresési próbát végeztem, amelyet megoldási próbálkozásként is hivatkozom. Mérésre került, hogy az 5 próba során mennyi esetben találta meg az eljárás az optimum megoldást.

6. táblázat - $1||C_{sum}$ tesztelés eredményeinek optimum találati gyakorisága

Optimumot megtaláló tesztek száma	Eredmény JAYA	Eredmény AMOJ
0	0	0
1	0	0
2	9	0
3	20	0
4	47	1
5	24	99

A 6. táblázat alapján látható, hogy az AMOJ eljárás hatékonyabban oldotta meg az ütemezési problémát. Csupán egyetlen feladatinstancia egyetlen megoldási próbálkozása esetén nem került az optimum érték megtalálásra. Minden más próbálkozás optimális megoldást eredményezett. Az AMOJ+FCMPGS módszer 99 feladatinstancia esetében öt futtatásból ötször talált optimumot, míg a JAYA+FCMPGS módszer csupán 24 feladatinstancia esetében tudott öt futtatásból ötször optimumot találni. Összevetve a JAYA operátor hatékonyságával az AMOJ-ban használt új módosító operátort erre az ütemezési problémára hatékonyabbnak tekintem.

A viszonylag magas megengedett iterációk száma (250000) miatt mérésre került az optimum érték megtalálásához szükséges lépések száma, hogy az algoritmusok hatékonyságát ebből a szempontból is megvizsgáljam és összehasonlítsam. A 7. és 8. táblázat mutatja be – nem egyenletesen elosztott – intervallumokra vetítve a végrehajtáshoz szükséges lépések eloszlását.

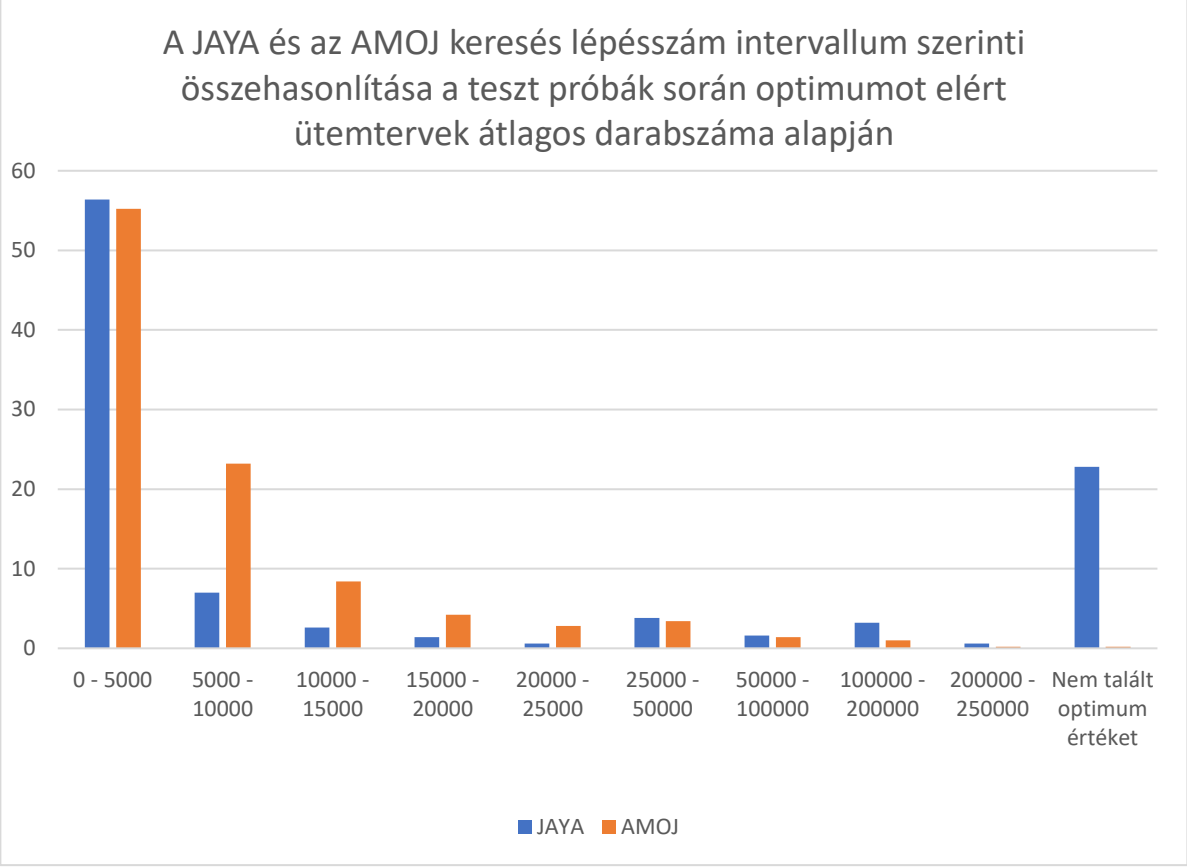
7. táblázat - Optimum megoldás megtalálásához szükséges iterációk számának eloszlása $1||C_{sum}$ tesztelés során JAYA módosító operátor alkalmazásával

Szükséges lépések száma	Teszt 1	2	3	4	5
0 – 5000	58	52	63	53	56
5000 – 10000	6	5	11	5	8
10000 – 15000	2	3	3	3	2
15000 – 20000	0	2	2	2	1
20000 – 25000	0	0	1	0	2
25000 – 50000	3	6	2	4	4
50000 – 100000	3	1	4	0	0
100000 – 200000	3	4	0	3	6
200000 – 250000	0	2	0	1	0
Nem talált optimumot	25	25	14	29	21

8. táblázat - Optimum megoldás megtalálásához szükséges iterációk számának eloszlása $1||C_{sum}$ tesztelés során AMOJ módosító operátor alkalmazásával

Szükséges lépések száma	Teszt 1	2	3	4	5
0 – 5000	57	57	52	56	54
5000 – 10000	20	26	26	23	21
10000 - 15000	7	9	13	4	9
15000 – 20000	5	3	3	5	5
20000 - 25000	3	1	2	5	3
25000 – 50000	4	2	3	4	4
50000 – 100000	2	0	1	2	2
100000 – 200000	1	2	0	1	1
200000 – 250000	0	0	0	0	1
Nem talált optimumot	1	0	0	0	0

A két algoritmus mért eredményeit a találati arány átlaga alapján szemlélteti a 14. ábra.



14. ábra - A JAYA és AMOJ eljárás hatékonyságának összehasonlítása

A mérés alapján megállapítható, hogy az AMOJ algoritmussal az esetek legalább 75%-ában az optimum érték megtalálásához kevesebb, mint 10000 iterációra volt szükség. Látható, hogy a 0 - 5000 lépésszám tartományban közel hasonló számú feladat esetén talált a két

algoritmus optimumot, azonban az 5000 – 10000 lépésszám tartományban az AMOJ eljárás jelentősen meghaladta az eredeti JAYA algoritmus teljesítményét.

Az AMOJ kereső a JAYA keresőhöz viszonyítva sokkal kevesebb iterációval (lépésszámmal) oldotta meg a legnehezebbnek tűnő feladatinstanciákat is ebben a problémakörben. A JAYA 200000-250000 iterációval tudta csak megoldani a legnehezebb instanciákat, míg az AMOJ ennek kevesebb mint a fele iterációszámmal oldotta meg a két legnehezebb feladatot. A többi feladat optimális megoldását jóval kevesebb iterációszámmal is elérte. Ezek a futási eredmények azt fejezik ki, hogy az AMOJ kereső sokkal hatékonyabb ezeknek a feladatoknak a megoldása során mint a JAYA, vagyis a saját fejlesztésű módosító operátor ezekben az esetekben kedvezőbb eredményeket ad, mint az eredeti JAYA módosító operátora.

Az AMOJ végrehajtásához szükséges futásidő eloszlásáról készült mérési eredményeket ismertetem a 12.1 mellékletben. Megállapítható, hogy az esetek 72 %-ban a szükséges futásidő nem haladja meg a 43 másodpercet, az esetek 89 %-ban az 1,5 percet. Csupán 4 feladatinstancia esetén volt szükség több, mint 3 perces futtatásra, de az algoritmus ezekben az esetekben is 5 perc alatt eredményt szolgáltatott.

6.1.3 Futási eredmények generált $1|d_i|L_{max}$ és $1|d_i|U_{sum}$ feladatok esetén

Az egygépes, határidővel rendelkező feladatok ütemezési problémáit is megvizsgáltam. Az $1|d_i|L_{max}$ és $1|d_i|U_{sum}$ feladatok csak célfüggvényben különböznek egymástól, ezért ezeket az ütemezési problémákat összevontan ismertetem. Mindkét feladat esetében ismert polinomiális időben optimum megoldást eredményező algoritmus. A problémák – célfüggvény nélküli – formális leírása az alábbi.

$\boldsymbol{J}$ — $\boldsymbol{J} = (j_1, j_2, \dots, j_i, \dots, j_n)$, n számú ütemezendő munka. A munkák egymástól függetlenül végrehajthatóak. Minden munkának egyetlen operációja van.

m — Egy gép áll rendelkezésre a munkák végrehajtására. A gép egyszerre csak egy munkát tud végrehajtani.

p_i — Az *i.* munka végrehajtásának idejét jelöli az *m* gépen.

d_i — Az *i.* munka teljesítésének (befejezésének) határidejét jelöli.

Célfüggvény $1|d_i|L_{max}$ esetén a legnagyobb késés (határidő túllépés) minimalizálása. $L_{max} = max_{i\epsilon \boldsymbol{J}}\,(C_i - d_i)$, ahol C_i jelöli az *i*-edik munka teljesítési idejét.

Célfüggvény $1|\,d_i\,|\,U_{sum}$ esetén a késő (határidőt túllépő) munkák darabszámának minimalizálása. $U_{sum} = \sum_{i\epsilon \boldsymbol{J}} \xi_i$, ahol ξ_i értéke 1 ha a munka túllépi az előírt d_i határidőt, egyébként az értéke 0. Formálisan:

$$\xi_i = \begin{cases} 1, & ha\ C_i > d_i, \\ 0, & egyébként. \end{cases} \tag{41}$$

Állítás: Az $1|d_i|L_{max}$ és $1|d_i|U_{sum}$ ütemezési probléma az RCMOMPSP modellre leképezhető.

Bizonyítás: Létezik olyan módszer, amely az $1|d_i|L_{max}$ és $1|d_i|U_{sum}$ ütemezési problémákat egyértelműen a RCMOMPSP problémára leképezi. A leképzés az alábbiak szerint végezhető el:

- Minden munka egy projektnek feleltethető meg. $\boldsymbol{P} := \boldsymbol{J}$
- Minden operáció egy egyedi feladatra képezhető le. A feladat ahhoz és csak ahhoz a projekthez tartozik, amely projektet ahhoz a munkához hoztunk létre, amelyhez a feladat kiindulásaként tekintett operációja tartozott.
  $\boldsymbol{T} := \boldsymbol{J}, \boldsymbol{TA} = \{(p_i, t_i)\} | 1 \leq i \leq n$
- A feladatok előfeltétel halmaza üres.
  $\boldsymbol{PRE}_j = \emptyset, \forall j \in \boldsymbol{T}$
- Egyetlen gép áll rendelkezésre, amelyet egyetlen erőforrás típusra lehet leképezni. Az így definiált erőforrástípusnak konstans egy egység a kapacitása.
  $$\boldsymbol{R} = \{r_1\}, \qquad RC_{r_1} = 1$$
- Minden feladat az r_1 erőforrástípust használja 1 kapacitással.
  $$rr_{j,1} = 1, \forall j \in \boldsymbol{T}$$
- A projektek lehetséges tulajdonságai közül csak a projekt határidőt alkalmazzuk.
  $\boldsymbol{PPROP} = \{P_{dTime}\}$.
- Minden egyes projekt határideje rendre megegyezik annak a munkának a határidejével, amelynek a projektet feleltettük meg.
  $\boldsymbol{PP}_i = \{d_i\}$, azaz $PP(p_i, P_{dTime}) = d_i$
- Az RCMOMPSP modellben csak egyetlen célfüggvény kerül meghatározásra, tetszőleges nem nulla fontosság súllyal.
- A célfüggvény $1|d_i|L_{max}$ ütemezési feladat esetén a projektek legnagyobb késési idejének minimalizálása. Az RCMOMPSP modellben a célfüggvény a definiált $f_{PL_{max}}$ célfüggvényre képezhető le.
- A célfüggvény $1|d_i|U_{sum}$ ütemezési feladat esetén a késedelmes projektek számának minimalizálása, amely az RCMOMPSP modellben a f_{PDC} függvénnyel definiált.

6.1.3.1 $1|d_i|L_{max}$ ütemezési feladat tesztelése

A tesztelés során 100 darab $1|d_i|L_{max}$ ütemezési feladat került generálásra. Minden ütemezési feladat 30 munkából állt. Az egyes munkák végrehajtási ideje egy véletlenszerűen választott egész szám az [1,100] intervallumból. Az r_{Cmax} egy véletlenszerűen választott egész szám a [0, C_{max}] intervallumból. Minden munka véletlenszerűen generált határideje az alábbi eljárás szerint került meghatározásra:

$$d_i = 1 + r_{Cmax} \,\%\, C_{max}. \tag{42}$$

Minden ütemezési feladat esetén az optimum érték kiszámításra került a „leghamarabbi határidejűt először” (*Earliset Due Date – EDD*) algoritmus alkalmazásával [68]. A leképzett RCMOMPSP problémára a hibrid megoldási megközelítést (MOSM+SC) alkalmazó

megoldóval generáltunk megoldásokat. A megoldás során alkalmazott keresési paramétereket a 9. táblázatban ismertetem.

9. táblázat - $1|d_i|L_{max}$ütemezési feladat tesztelési beállításai

Paraméter	Érték
Keresési iterációk száma	25000
Populáció mérete	2
Ismételt teszt próbák száma	5

A teszt során mind a 100 generált probléma megoldásra került. Az MOSM+SC hibrid eljárás pontosan ötször került futtatásra minden példány esetén, mely során a kereső minden esetben új, véletlenszerűen generált kezdeti populációt generált. A futási eredmények összevetését az EDD garantáltan optimumot adó megoldásával mutatja a 10. táblázat.

10. táblázat - $1|d_i|L_{max}$ futási eredményeinek áttekintő összegzése

Kiértékelési nézőpont	Érték
Probléma példányok száma	100
Megtalált optimális ütemtervek száma	100
Találati hatékonyság	1.0
Átlagos relatív távolság az optimum értéktől	0
Maximális relatív távolság az optimum értéktől	0

Az MOSM+SC hibrid megoldó eljárás a teszt ciklus során az ötödik próba után minden ütemezési probléma esetén megtalálta az optimum megoldást. Mérésre került, hogy az 5 teszt során mennyi esetben találta meg az eljárás az optimum megoldást.

11. táblázat - $1|d_i|L_{max}$ tesztelés eredményeinek optimum találati gyakorisága

Optimumot megtaláló tesztek száma	Eredmény
0	0
1	0
2	0
3	5
4	8
5	87

Kijelenthető, hogy az MOSM+SC hibrid módszer a vizsgált problémainstanciák 87%-ában minden egyes futtatás esetén megtalálta az optimumot. Az instanciák 8%-ában öt futtatásból négyszer, míg az instanciák 5%-ában öt futtatásból legalább háromszor optimális megoldást talált.

6.1.3.2 $1|d_i|U_{sum}$ ütemezési feladat tesztelése

A tesztelés során 100 darab $1|d_i|U_{sum}$ ütemezési feladat került generálásra. Minden ütemezési feladat 30 munkából állt. Az egyes munkák végrehajtási ideje egy egyenletes eloszlás szerint véletlenszerűen választott egész szám az [1,100] intervallumból. Minden munka generált határideje az alábbi képlet szerint került meghatározásra:

$$d_i = \frac{(\sum_{m<i} p_m)}{4}. \quad (43)$$

Minden munka határideje megegyezik a kisebb azonosítójú munkák műveleti idejének összegének negyedével. A számítás célja, hogy a határidők még optimális ütemezésnél is sérüljenek, azaz a késő munkák darabszáma viszonylag nagy legyen.

Minden ütemezési feladat esetén az optimum érték kiszámításra került a Moore algoritmussal [68]. A leképzett RCMOMPSP problémára az MOSM+SC hibrid megoldási megközelítést alkalmazó megoldóval generáltunk megoldásokat. A megoldás során alkalmazott keresési paramétereket a 12. táblázatban ismertetem.

12. táblázat - 1|d_i|U_{sum} ütemezési feladat tesztelési beállításai

Paraméter	Érték
Keresési iterációk száma	25000
Populáció mérete	2
Ismételt teszt próbák száma	5

Minden tesztelési ciklusban mind a 100 generált probléma megoldásra került. Az MOSM+SC hibrid eljárás pontosan ötször került futtatásra minden példány esetén, mely során a kereső minden esetben új, véletlenszerűen generált kezdeti populációt generált. A futási eredmények összevetését a Moore garantáltan optimumot adó megoldásával mutatja a 13. táblázat.

13. táblázat - 1|d_i|U_{sum} futási eredményeinek áttekintő összegzése

Kiértékelési nézőpont	Érték
Probléma példányok száma	100
Megtalált optimális ütemtervek száma	100
Találati hatékonyság	1.0
Átlagos relatív távolság az optimum értéktől	0
Maximális relatív távolság az optimum értéktől	0

Ebben a tesztelésben is egy ütemezési probléma esetén 5 megoldási próbálkozást tettem. Az MOSM+SC hibrid megoldó eljárás a teszt ciklus során az ötödik teszt után minden ütemezési probléma esetén megtalálta az optimum megoldást. Mérésre került, hogy a problémát megoldó 5 teszt során mennyi esetben találta meg az eljárás az optimum megoldást.

14. táblázat - 1|d_i|U_{sum} tesztelés eredményeinek optimum találati gyakorisága

Optimumot megtaláló tesztek száma	Eredmény
0	0
1	1
2	2
3	3
4	8
5	86

Látható, hogy a vizsgált esetek 86%-ában az MOSM+SC algoritmus 100%-os hatékonysággal ötből ötször) megtalálta az ütemezési feladat optimális megoldását. Még a legnehezebbnek tűnő feladatinstancia esetében is öt próbálkozásból legalább kétszer optimumot talált. Ezek az eredmények alátámasztják a javasolt módszer hatékonyságát, mivel a véletlenszerűen választott kiindulási megoldástól függetlenül megtalálja a kedvező megoldásokat.

6.1.4 Futási eredmények $1|prec, d_i|L_{max}$ benchmark feladatokon

Az $1|\text{prec}, d_i|L_{max}$ ütemezési feladat a 6.1.3 alfejezetben ismertetett $1|d_i|L_{max}$ ütemezési feladat kiterjesztésének tekinthető, így ebben az alfejezetben a feladat leírásánál csak a kiterjesztés elemeit ismertetem, illetve a kiterjesztés leképzését az RCMOMPSP modellre.

Az $1|\text{prec}, d_i|L_{max}$ feladat esetén a munkák végrehajtása egymástól nem független, a feladatok előfeltételét a prec szimbólummal jelzett előfeltétel (*precedencia*) gráf írja le.

A 6.1.3 alfejezetben ismertetett feladatleírást így az alábbiak szerint módosítom:

$\boldsymbol{J}$ $\boldsymbol{J} = (j_1, j_2, \dots, j_i, \dots, j_n)$, n számú ütemezendő munka. A munkák végrehajtása egymástól nem független.

$prec$ $prec = \{(j_m, j_n)\}$ halmaz eleme akkor, ha a j_m munka előfeltétele a j_n munka végrehajtásának. j_n munka végrehajtása nem kezdődhet meg, amíg j_m munka nem került végrehajtásra. A $prec$ halmaz által leírt irányított gráf körútmentes.

Tesztelés:

A tesztelési feladatokat Kolisch és Hartmann által publikált, széles körben elfogadott és alkalmazott PSLIB benchmark feladatok alapján hoztam létre [17]. Az online elérhető PSLIB RCPSP (single mode) J30, J60 és J120 feladatsorokat a teszt során leképeztem az $1|\text{prec}, d_i|L_{max}$ feladatra. A J30, J60 és J120 adatok rendre 30, 60 és 120 munkát tartalmaznak. A munkák meghatározott végrehajtási időit, előfeltételeit átvettem, a feladat erőforrás korlátait 1 gépes környezetre módosítottam. Ezen túlmenően minden egyes munkához egy d_i határidőt generáltam, amelyet a következő alfejezetekben mutatok be.

Az így kapott ütemezési feladatokat megoldottam a Lawler algoritmussal [68], hogy az optimális ütemezés célfüggvény értéke a saját algoritmusom hatékonyságának kiértékeléséhez rendelkezésre álljon.

6.1.4.1 $1|\text{prec}, d_i|L_{max}$ feladat CPM módszerrel generált határidőkkel

Teljesítési határidők generálásához a feladatot előbb leképeztem egy erőforráskorlát mentes ütemezési problémára és azt kritikus út módszerrel (*Critical Path Method – CPM*) oldottam meg [70]. Az erőforráskorlát mentes ütemezési feladat esetén minden feladat azonnal végrehajtható, amennyiben a feladat előfeltétel feladatai végrehajtásra kerültek. Ezzel a módszerrel minden feladat (munka) esetén meg lehet határozni a lehetséges legkorábbi kezdési

időpontot (*Earliest Start Time – EST*) és legkorábbi befejezési időpontot (*Earliest Finishing Time – EFT*).

A tesztfeladatokban minden j_i feladat (munka) esetén a d_i értéket a CPM által meghatározott lehetséges legkorábbi kezdési időpont szerint állítottam be:

$$d_i = EST_i \ . \tag{44}$$

Ennek az ütemezési problémának a jellegéből adódóan így minden problémainstancia esetében legalább egy feladat (munka) késedelmes lesz (határidő után fejeződik be).

Az így előállított RCMOMPSP problémát az MOSM+SC hibrid megoldási módszerrel oldottam meg. A megoldás során alkalmazott keresési paramétereket a 15. táblázatban ismertetem.

15. táblázat - 1|prec, $d_i|L_{max}$ ütemezési feladat tesztelési beállításai CPM EST határidő korláttal

Paraméter	Érték
Keresési iterációk száma	25000
Populáció mérete	2
Ismételt teszt próbák száma	5

A tesztelt feladatok számát, az optimum megoldást találó ütemezések számát a 16. táblázatban ismertetem.

16. táblázat - 1|prec, $d_i|L_{max}$ ütemezési feladat tesztelési eredményei CPM EST határidő korláttal

Kiértékelési nézőpont	J30	J60	J120
Ütemezési feladatok száma	480	480	600
Optimum megoldást megtaláló esetek száma	480	480	600
Optimum elérésének találati aránya	1	1	1
Optimumtól mért átlagos távolság	0	0	0
Optimumtól mért maximális eltérés	0	0	0

Látható, hogy az alkalmazott MOSM+SC hibrid megoldó módszer minden feladatinstancia esetében megtalálta az optimális ütemezést. A 17. táblázat mutatja be az optimum elérésének gyakoriságát.

17. táblázat - 1|prec, $d_i|L_{max}$ ütemezési feladat optimum találati gyakorisága CPM EST határidő korláttal

Optimumot megtaláló tesztek száma	J30	J60	J120
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	480	480	600

Az eredményt kiemelkedően jónak ítélem, hiszen minden feladatpéldány esetén, minden próbálkozás során megtalálta az eljárás az optimum értéket. A további vizsgálatok érdekében egy másik határidő generálási eljárást is megvizsgáltam.

6.1.4.2 1|prec, $d_i|L_{max}$ feladat generált, felső korlátos határidőkkel

Teljesítési határidők generálásához kiszámítottam a végrehajtási idők összegét. Ezt követően minden feladat esetén egy véletlenszerű számot választottam a $[0, \sum_{\forall i \in J} p_i]$ intervallumból. A leképzett RCMOMPSP problémára az MOSM+SC hibrid megoldási megközelítést alkalmaztam. A megoldás során alkalmazott keresési paramétereket a 18. táblázatban ismertetem.

18. táblázat - 1|prec, $d_i|L_{max}$ ütemezési feladat tesztelési beállításai felső korláttal rendelkező, véletlenszerűen generált határidő korláttal

Paraméter	Érték
Keresési iterációk száma	25000
Populáció mérete	2
Ismételt teszt próbák száma	5

A tesztelt feladatok számát, az optimum megoldást találó ütemezések számát a 19. táblázatban ismertetem.

19. táblázat - 1|prec, $d_i|L_{max}$ ütemezési feladat tesztelési eredményei felső korláttal rendelkező, véletlenszerűen generált határidő korláttal

Kiértékelési nézőpont	J30	J60	J120
Ütemezési feladatok száma	480	480	600
Optimum megoldást megtaláló esetek száma	480	480	600
Optimum elérésének találati aránya	1	1	1
Optimumtól mért átlagos távolság	0	0	0
Optimumtól mért maximális eltérés	0	0	0

A 20. táblázatban ismertetem, hogy a teszt ciklus során végrehajtott 5 teszt esetén milyen eloszlás szerint találta meg az eljárás az ismert optimumot.

20. táblázat - 1|prec, $d_i|L_{max}$ ütemezési feladat optimum találati gyakorisága felső korláttal rendelkező, véletlenszerűen generált határidő korláttal

Optimumot megtaláló tesztek száma	J30	J60	J120
0	0	0	0
1	0	0	0
2	0	0	1
3	0	0	0
4	0	0	3
5	480	480	596

Látható, hogy a J30 és J60 sorozatok mind a 480 problémapéldányának esetében öt futtatásból ötször ért el a javasolt módszer optimális megoldást. Ez az eredmény nagyon kiemelkedőnek tekinthető és az előzetes várakozásomat jelentősen meghaladta.

A J120 sorozat esetében is nagyon kiváló eredményeket ért el a javasolt módszer. A 600 problémapéldányból 596 esetében öt futtatásból ötször találta meg az optimumot. 3 példány esetében ötből négyszer talált optimumot és csupán egyetlen problémapéldány esetében fordult elő, hogy öt futtatásból kétszer talált optimumot.

Az elért nagyon kiváló eredmények igazolták, hogy nagyméretű feladatok esetén is nagyon hatékonyan működik a javasolt megoldási módszer.

6.2 Futási eredmények PSLIB RCPSP benchmark feladatokon

Kolisch és Hartmann publikált széles körben elfogadott és alkalmazott projektütemezési benchmark feladatokat [17]. A PSPLIB problémagyűjtemény nem többmódozatú (*single mode*) RCPSP feladatait az alábbiak szerint lehet összefoglalni:

Munkák Adott a munkák egy előre megadott halmaza:

$\boldsymbol{J} = (j_{vs}, j_1, j_2, \dots, j_i \dots, j_n, j_{vt})$.

Minden benchmark feladat tartalmaz két extra virtuális munkát, egy virtuális kezdő munkát (j_{vs}) és egy virtuális zárómunkát (j_{vt}). A virtuális munkák végrehajtási ideje nulla, szerepük a független feladatok kezdetének és a teljes feladat befejezésének szinkronizálása.

Követő munkák Minden j feladat rendelkezhet követő feladatokkal, amelyek végrehajtása nem kezdődhet hamarabb, mint a j feladat végrehajtási ideje.

Erőforrások Adott az állandó kapacitású erőforrások halmaza, amelyek a munkák végrehajtásához szükségesek. Az erőforrások kapacitása adott és véges.

Feldolgozási idők Minden munka esetén adott a szükséges erőforrások kapacitásigénye és a feldolgozáshoz szükséges idő.

Állítás: Az ismertetett RCPSP modell a kiterjesztett RCMOMPSP modellre leképezhető.

Bizonyítás: Létezik olyan módszer, amely az RCPSP ütemezési problémákat egyértelműen a RCMOMPSP problémára leképezi. A legképzés az alábbi módon végezhető el.

- Egy projekt kerül definiálásra: $\boldsymbol{P} = \{p_1\}$.
- Minden munka pontosan egy feladatnak feleltethető meg. Minden feladat ugyanahhoz a projekthez (p_1) tartozik: $\boldsymbol{T} = Jobs, \boldsymbol{TA} = \{(p_1, t_j)\} | \forall t_j \in \boldsymbol{T}$.
- Minden erőforrás egy erőforrástípusnak feleltethető meg. Az erőforrás előre definiált kapacitáskorlátja rendre a létrehozott erőforrástípus kapacitásával egyenlő: $\boldsymbol{R} = Erőforrások;\ rendre\ RC_{r_1}, RC_{r_2}, \dots\ RC_{r_{NK}}$.
- A munkák erőforrásigénye rendre a projekt feladatok erőforrástípus szükségleteivel egyeznek meg.
- Ha egy munka rendelkezik definiált követő (*successor*) munkával, akkor az RCMOMPSP modellben a követő munka alapján létrehozott projektfeladat (*task*) előfeltétel (*predecessor*) feladatként tekinti a munkához létrehozott projektfeladatot (task-ot).
- A projektütemezési feladat csak egy célfüggvényt tartalmaz; az utolsó projektfeladat befejezési idejének a minimalizálását: $f_{PC_{max}}$.

6.2.1 Futtatási eredmények AMOJ eljárás használatával

A tesztelés során az AMOJ metaheurisztikus keresőt alkalmaztam az FCMPGS generálási sémával. A keresési paramétereket mutatja be a 21. táblázat.

21. táblázat - PSLIB benchmark feladatok tesztelési beállításai AMOJ eljárás használatával

Paraméter	Érték
Keresési iterációk száma	1000
Populáció mérete	100
Ismételt teszt futtatások száma	5

Az első tesztelési ciklusban a PSLIB benchmark feladatok J30-as adathalmazát használtam. Ebben a példahalmazban minden projekt 30 munkát tartalmaz a 2 virtuális munkán kívül. Minden ütemezési feladat esetén 4 kapacitáskorlátos erőforrás áll rendelkezésre. A második tesztelési ciklusban a J60-as adathalmazt használtam, amely már ütemezési feladatonként 60 munkát és 2 virtuális munkát tartalmaz.

Mindkét tesztelési ciklus esetén a hibrid megoldó módszer összesen típusonként 480 benchmark feladatot oldott meg. Minden ütemezési feladatot pontosan 5 alkalommal oldottam meg. Ennek egyik oka az volt, hogy a keresés során véletlenszerűen választ keresési kezdőpontot a módszer. A másik ok az volt, hogy a keresés során véletlenszerűen választott módosításokat hajt végre az algoritmus. Ezek együtt hatással vannak a végeredményre, melyek indokolják többszöri futtatások elvégzését. A tesztek futtatása után a kapott eredményeket összevetettem a PSLIB honlapján közölt ismert legjobb megoldások adataival. Az eredményeket a 22. táblázatban ismertetem.

22. táblázat - PSLIB benchmark futási eredményeinek áttekintő összegzése AMOJ eljárás használatával

Kiértékelési nézőpont	J30 adathalmaz	J60 adathalmaz
Ütemezési feladatok száma	480	480
Ismert legjobb megoldást megtaláló esetek száma	447	350
Az ismert legjobb megoldás elérő találati arány	0,93125	0,729166667
Az ismert legjobb megoldástól mért átlagos relatív távolság	0,001109575	0,009709142
Az ismert legjobb megoldástól mért maximális relatív eltérés	0,044776119	0,090909091

A 22. táblázat adatai alapján megállapítható, hogy a 30 munkát tartalmazó feladatpéldányok ~93%-ában az ismert legjobb megoldást szolgáltatta a kereső, míg 60 munkát tartalmazó feladatpéldányok esetén ez az arány ~73%.

Mivel az összesített eredményeket mutató táblázat az 5 tesztelési próbálkozás együttes eredményét mutatja, ezért részletesebb képet ad a 23. táblázat. Ebben a részletesebb táblázatban az ismert legjobb megoldásokat megtaláló futtatások szerint tüntetem fel az eredményeket.

23. táblázat - PSLIB benchmark ismert legjobb megoldás találati gyakorisága AMOJ eljárás használatával

Ismert legjobb megoldást elérő futtatások száma	J30	J60
0	33	130
1	8	3
2	14	1
3	10	4
4	9	5
5	406	337

Annak ellenére, hogy a hibrid kereső a feladat jellegéről semmilyen problémaspecifikus információt nem vett figyelembe, az ismert optimum elérését az esetek nagy részében mind az 5 véletlenszerű indítás esetén elérte. Ez az AMOJ algoritmusban használt új egyed módosító operáció jó teljesítményét és az abból eredő új problématér-bejárási stratégia hatékonyságát és rugalmasságát mutatja.

J30-as adathalmaz vizsgálata során 406 probléma esetén az ismert optimum öt próbálkozásból ötször megtalálásra került. 33 problémapéldány esetében nem talált optimumot. A 33 esetben a kereső által szolgáltatott ütemterv teljesítménymutatója nagyon kis eltérést mutatott az optimumtól: 1 időegység 30 probléma esetén, 2 időegység 2 probléma esetén és 3 időegység 1 probléma esetén. A legrosszabb esetben az optimum érték 67, az ütemező által szolgáltatott 70-es érték abszolút távolsága 3 időegység, míg relatív távolsága 0,0447, amely megítélésem szerint a gyakorlati alkalmazások esetében messze túlteljesíti az alkalmazható kvázi-optimum feltételeit.

J60-as tesztfeladatok esetén az elért eredmények kismértékben elmaradnak a J30-as tesztsor eredményeitől. Ez egyrészt a sokkal nagyobb keresési térre, másrészt a probléma mérete ellenére sem növelt keresési paraméterekre vezethető vissza. A feladatcsoport esetében 337 esetben találta meg az algoritmus mind az 5 indított keresés esetén az ismert legjobb megoldást. 130 feladat esetén az algoritmus nem találta meg az ismert legjobb megoldást, de ezekben az esetekben is nagyon jól közelítette azt. A legrosszabb esetben az ismert legjobb és a megtalált megoldás között 10 időegység különbség áll fenn, amely relatív távolságban 0,0909 értéknek felel meg. Ezt az eltérést – a J30 esethez hasonlóan – a gyakorlati alkalmazásokban kvázi-optimumot teljesítőnek ítélem meg.

6.2.2 Futtatási eredmények MOSM eljárás használatával

Az AMOJ eljáráshoz hasonlóan az MOSM használatával is végeztem méréseket a PSLIB benchmark feladatokon. A kereső eljárás során a 24. táblázatban ismertetett paramétereket használtam.

24. táblázat - Az MOSM eljárás keresési paraméterei a PSLIB benchmark feladathalmaz J30, J60 és J120 feladatai esetén

Paraméter	**Érték**
Generációk száma (NG)	500, 2500 és 25000
Populáció mérete (NM)	2
Ismételt teszt futtatások száma	15

Az MOSM algoritmus eredményeit más kutatók által publikált megoldó eljárások eredményeihez hasonlítottam. A vizsgálatok elvégzéséhez minden egyes p PSLIB RCPSP benchmark ütemezési feladat esetén az ismert alsó korlát (lower bound – LB_p) került figyelembevételre. PSLIB J30-as feladatai esetén publikáltak az ellenőrzött módon kiszámított és ismert optimum LB_p értékek. A PSLIB J60 és J120 feladatok esetén is a szakirodalomban javasolt mintát követtem. Az erőforráskorlát nélküli, kritikus út megoldó módszer (*Critical Path Method – CPM*) által számított legkorábbi projektbefejezési időpontot vettem figyelembe LB_p alsó korlátként.

Az ismert LB_p értéket összehasonlítási alapot adó referenciaértéknek tekintettem a célfüggvény szempontjából. A többi kutatóhoz hasonlóan minden probléma esetén 15 futtatást végeztem. A 15 próbálkozás után elért legjobb célfüggvény értéket a $C_{best,p}$ jelöli egy adott p benchmark feladatinstancia esetén. Ezt követően meghatározásra került az átlagos relatív eltérés (*Average Relative Deviation – ARD*) értéke az alábbi képlet alkalmazásával:

$$ARD = \frac{\sum_{p=1}^{P} \frac{C_{best,p} - LB_p}{LB_p}}{P} 100\ [\%]\ . \tag{45}$$

Az ARD értékek mind a J30, J60 és J120 problémákra kiszámításra kerültek. Az irodalomban összehasonlítási referenciaként szolgáló publikációkban rendre 1000, 5000 és 50000 célfüggvény kiértékelési lépés kerül alkalmazásra, így a mérés során a célfüggvények kiértékelésének maximális számát hasonlóan korlátoztam. A 25., 26. és 27. táblázatban más publikációkban közölt eredményekkel került az MOSM eljárás összehasonlításra.

25. táblázat - Átlagos relatív eltérés összehasonlítása J30 feladatok esetén MOSM eljárás használatával

Algoritmus	ADR érték - célfüggvény kiértékelésének maximális száma szerint		
	1000	5000	50000
MOSM (saját módszer)	**0,12**	**0,03**	**0,003**
QIGA (2021) [71]	0,20	0,12	0,06
TS-MODE (2020) [71]	0,06	0,01	0,00
HGA (2008) [72]	0,27	0,06	0,02
GRASP-FBI-SS (2013) [73]	0,57	0,39	0,23
Sequential(SS(FBI)) (2018) [74]	0,10	0,02	0,00
EQIGA (2022) [75]	0,06	0,02	0,00
Memetic algorithm (2020) [76]	-	0,00	0,00
JPSO (2011) [77]	0,29	0,14	0,04

26. táblázat - Átlagos relatív eltérés összehasonlítása J60 feladatok esetén MOSM eljárás használatával

Algoritmus	ADR érték - célfüggvény kiértékelésének maximális száma szerint		
	1000	5000	50000
MOSM (saját módszer)	**12,046**	**11,42**	**10,97**
QIGA (2021) [71]	12,36	12,10	11,77
TS-MODE (2020) [71]	11,90	11,21	10,629
HGA (2008) [72]	11,56	11,10	10,7
GRASP-FBI-SS (2013) [73]	12,88	12,42	11,96
Sequential(SS(FBI)) (2018) [74]	11,38	10,93	10,58
EQIGA (2022) [75]	11,70	11,22	10,74
Memetic algorithm (2020) [76]	-	10,72	10,55
JPSO (2011) [77]	12,03	11,43	11,00

27. táblázat - Átlagos relatív eltérés összehasonlítása J120 feladatok esetén MOSM eljárás használatával

Algoritmus	ADR érték - célfüggvény kiértékelésének maximális száma szerint		
	1000	5000	50000
MOSM (saját módszer)	**38,03**	**34,16**	**33,94**
QIGA (2021) [71]	36,30	36,02	35,08
TS-MODE (2020) [71]	34,40	32,86	30,59
HGA (2008) [72]	34,07	32,54	31,24
GRASP-FBI-SS (2013) [73]	38,16	37,30	36,32
Sequential(SS(FBI)) (2018) [74]	34,01	32,52	31,16
EQIGA (2022) [75]	36,22	35,12	33,55
Memetic algorithm (2020) [76]	-	32,76	31,12
JPSO (2011) [77]	35,71	33,88	32,89

J30-as feladatok esetén az ARD átlagos relatív eltérés 0,003% ha az 50000 kiértékelési korlátot tekintem. Ez azt jelenti, hogy a vizsgált 480 ütemezési feladat esetén az MOSM eljárás 479 esetben megtalálta az optimális ütemezést. A J60 benchmark feladatok esetén az MOSM

eljárás ARD átlagos relatív eltérése a legjobban teljesítő algoritmushoz képest csupán 0,42, ha az 50000 lehetséges célfüggvény kiértékelést tekintem. J120 feladatok esetén az MOSM által elért ARD csupán 3.35 értékkel haladta meg a legjobb értéket.

Ismételten kiemelem, hogy az MOSM módszerem nem dedikáltan az RCPSP problémára, nem a Cmax célfüggvényre és nem a PSLIB benchmark feladatok megoldására van tervezve és optimalizálva, hanem sokkal általánosabb célú felhasználást tesz lehetővé. Az itt bemutatott tesztproblémákat egy szélesebb és bonyolultabb problématerület egyik speciális eseteként oldottam meg. Nem használtam semmilyen problémaspecifikus jellemzőt a megoldás létrehozásakor és nem használtam semmilyen empirikus konstanst a teljesítmény javítására. Ezeket figyelembe véve az elért eredményeket kiemelkedően jónak tekintem, mivel figyelemre méltóan közel állnak a legjobb eredményekhez, sőt, sok más eredménynél jobbak is. Az alfejezetben bemutatott eredményeket az [S18] folyóiratcikkben publikáltam.

Az MOSM algoritmus PSLIB J30-as benchmark feladatok végrehajtásához szükséges futásidő eloszlásáról készült mérési eredményeket ismertetem a 12.2 mellékletben. Megállapítható, hogy az esetek túlnyomó többségében a futásidő nem haladja meg a 0,5 másodpercet és a leghosszabb végrehajtási időt igénylő esetben sem több, mint 2,3 másodperc.

6.3 Többprojektes, több célfüggvényt alkalmazó feladatok vizsgálata

A harmadik fő tesztelési megközelítés során az új modellt és a megoldó eljárást vizsgáltam több projekt együttes kezelése és az ütemterv kialakítása közben több figyelembe vett célfüggvény szerint. A végrehajtott tesztek csoportosítva, egymásra épülő alfejezetekbe szervezve kerülnek bemutatásra. A lépésről lépésre bemutatás szerepe a végrehajtott tesztek fő bemeneti paraméterei, a tesztek eredményei és az eredmények értékelő ismertetése mellett az, hogy betekintést nyújtson a tesztelés során azonosított további következtetések megállapításának hátterébe.

6.3.1 Többprojektes, több célfüggvény szerinti feladat

A több projektet tartalmazó tesztfeladatot az RCPSP benchmark feladatok alapján készítettem el. A PSLIB30 feladatok első 30 benchmark feladatát együttesen tekintettem egy projekt portfóliónak. Az így kapott többprojektes feladatokon további, az alábbiakban ismertetett kiegészítéseket is végrehajtottam.

A fejezetben alkalmazott célfüggvények rendre: $L_{max}, T_{max}, T_{sum}, U_{sum}, C_{max}, C_{sum}$. Minden célfüggvény esetén a minimalizálás az optimalizálási cél. A vizsgált célfüggvények az RCMOMPSP modellben a 28. táblázat szerint vannak definiálva.

28. táblázat - A többprojektes tesztsorozatok célfüggvények meghatározása

Tesztelési célfüggvény	**Célfüggvény neve**	**Definíció**
L_{max}	A legnagyobb késés	$f_{PC_{max}} = max_{t_j \in \boldsymbol{T}} (TL(t_j))$
T_{max}	A legnagyobb csúszás	$f_{TT_{max}} = max_{t_j \in \boldsymbol{T}} \left(TT(t_j)\right)$

T_{sum}	Csúszások összege	$f_{TT_{sum}} = \sum_{t_j \epsilon \boldsymbol{T}} (TT(t_j))$
U_{sum}	Késő feladatok száma	$f_{TDC} = \sum_{t_j \in \boldsymbol{T}} \lambda_{tdelayed}(t_j)$
C_{max}	A legkésőbbi befejezési időpont	$C_{max} = max_{t_j \in T}(C_j)$
C_{sum}	Az összes feladat befejezési idejének összege	$C_{sum} = \sum_{t_j \in \boldsymbol{T}} (C_j)$

A kísérleti futtatások során megoldási módszerként az MOSM + SC hibrid módszert alkalmaztam. Az egyes tesztek összehasonlíthatósága érdekében minden teszt ugyanolyan kezdeti paramétereket és állapotteret használ. A keresés során a kiindulási feladatprioritás-vektor a párhuzamos generálási séma alkalmazásával kerül meghatározásra. A generálási séma a legkorábbi befejezési időpont prioritási szabályát használta.

29. táblázat - Többprojektes PSLIB benchmarkon alapuló projekt portfólió feladatok tesztelési beállításai

Paraméter	Érték
Iterációk száma	25000
Populáció mérete	2

6.3.2 Egyedi célfüggvények szerinti vizsgálat

Az első tesztelési csoportban a többprojektes ütemezési feladatot előre kiválasztott projektek határidőinek megadásával egészítettem ki. Minden ötödik projekt esetén (5., 10., 15. és 20.) projekt teljesítési határidőt határoztam meg (P_{dTime}). A PSLIB benchmark feladatok definíciója miatt a projekt szinten meghatározott teljesítési határidő megadása lehetséges azáltal is, hogy a projektek utolsó, virtuális zárófeladatának adunk meg teljesítési határidőt. A kiválasztott projektek esetén a PSLIB benchmark feladatban dokumentált legjobb C_{max} értékek alapján határoztam meg a projekt teljesítési határidejét. A kiválasztott projekt teljesítési határideje megegyezik a kisebb számú kiválasztott projektek C_{max} érték szerinti összegével, azaz:

$$P_{dTime,i} = \sum_{j \leq i} C_{max,j} \mid i,j \in \{5,10,15,20) \qquad (46)$$

Az első tesztcsoportban végrehajtott tesztek során mindig rendre egyetlen célfüggvény kapott nem nulla prioritást. A végrehajtott tesztek elsődleges vizsgálati célja, hogy ellenőrizzem a kereső vezérelhetőségét egyetlen kitüntetett célfüggvény szerint. A tesztek másodlagos vizsgálati célja az, hogy az egyedi cél szerint keresett célfüggvény értékeket a későbbi tesztek eredményeinek kiértékelésénél referencia értékként lehessen alkalmazni.

A 30. táblázatban ismertetem az 1-6. teszt célfüggvény prioritásait, valamint a keresés által adott megoldás kiértékelését az egyes célfüggvények tekintetében.

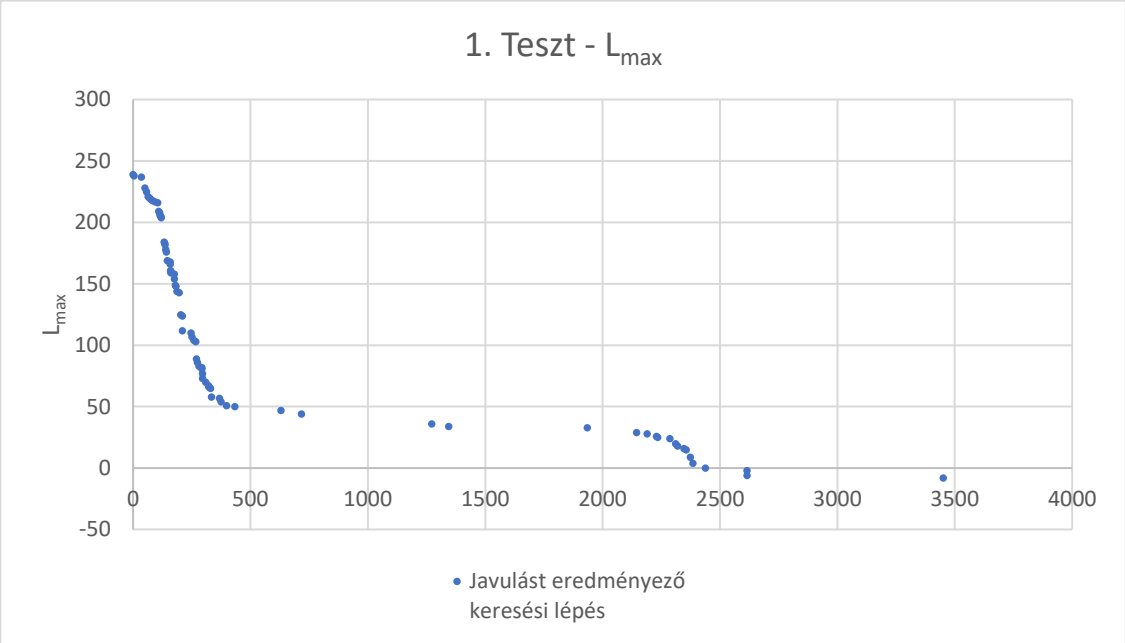
30. táblázat - Egyedi célfüggvény értékek szerint végrehajtott tesztek eredménye

Teszt	Célfüggvény prioritása	Célfüggvény értéke

	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
1	1	0	0	0	0	0	-8	0	0	0	321	79707
2	0	1	0	0	0	0	0	0	0	0	302	79557
3	0	0	1	0	0	0	0	0	0	0	311	80470
4	0	0	0	1	0	0	256	256	436	2	323	84605
5	0	0	0	0	1	0	237	237	637	4	276	79837
6	0	0	0	0	0	1	227	227	548	4	289	72393

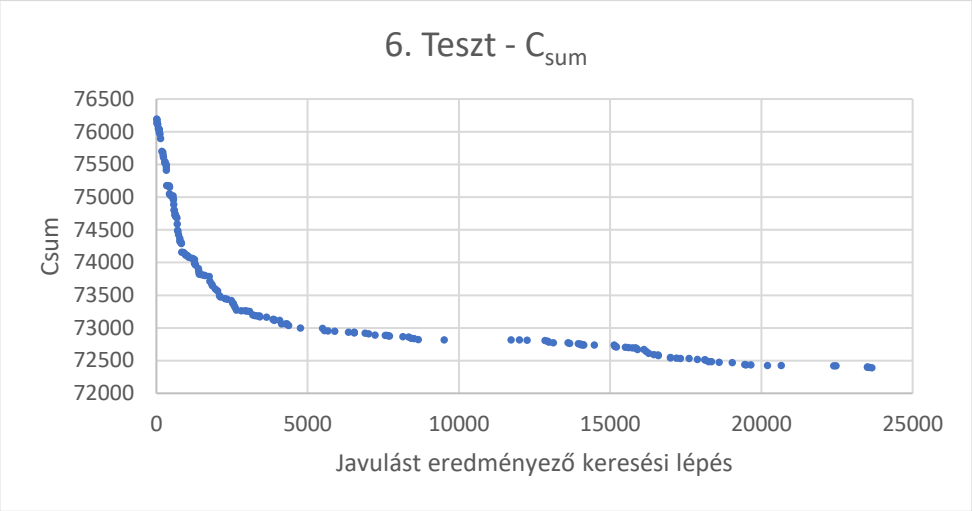
Megvizsgáltam a kereső teljesítményét, hogy milyen karakterisztikával közelít a kereső által megtalált legjobb eredményhez. A 11. függelékben részletesen megtekinthetőek a teszt végrehajtása során elért részeredmények. A függelékben csatolt, az algoritmus végrehajtási lépéseit rögzítő táblázatokban nem minden iterációs lépés kerül feljegyzésre. A futási eredmények táblázatában csak akkor kerül egy új adatsor rögzítésre, amikor a nem nulla súllyal meghatározott célfüggvény szerint a keresés javulást ér el. Két lépés között a kereső által megvizsgált esetek rosszabb eredményt adhatnak. A kereső működésének karakterisztikáját az 1. és az 6. teszt végrehajtásának kiértékelésével ismertetem. A további tesztek karakterisztikája a függelékben tekinthető meg.

A 15. ábraábrán látható az 1. teszt végrehajtásának keresési karakterisztikája. Látható, hogy a kereső az első 500 iterációban gyakran talál az addig ismert megoldástól jobb ütemezést. Ezt követően a jobb megoldások megtalálásához szükséges iterációk száma nő. Megfigyelhető, hogy a 2000-2500 iterációs tartományban ismét több javítást tud elvégezni. A bemutatott 15. ábra igazolja, hogy a kereső egy feltételezett és elvárható lecsengési görbét követ.



15. ábra - Kereső megoldást találó karakterisztikájának vizsgálata L_{max} célfüggvény esetén

A 16. ábra ismerteti a 6. teszt végrehajtásának karakterisztikáját. Ismét megfigyelhető, hogy a kereső a keresés kezdeti fázisában gyakran és nagy léptékű javításokat talál, majd a javítás gyakorisága és mértéke csökken. Ez megfelel az elvárásainknak, hiszen egyre kisebb valószínűséggel tud az MOSM kereső jobb kombinációt találni.



16. ábra - Kereső teljesítményének vizsgálata C_{sum} célfüggvény esetén

A 31. táblázat tartalma megegyezik a 30. táblázat tartalmával, azonban az elemzés segítése miatt kiemeltem az egyes tesztek esetén a nem negatív súllyal alkalmazott célfüggvényt, valamint az összes mért eredmény esetén az egyik minimális értéket. Jól látható, hogy az egyes célfüggvények eredményei és a célfüggvények számára megadott prioritások között összefüggés van. Azonban ez nem minden célfüggvény esetén igaz, mivel az U_{sum} esetén más célfüggvény számára megadott súllyal jobb eredményt talált a kereső. Ennek feltételezett oka, hogy az U_{sum} célfüggvény értékkészlete kicsi és a kereső így nem tud hatékonyan javító megoldásokat keresni. Ennek tovább elemző vizsgálatát a harmadik tesztcsoportban végeztem el és a követő vizsgálat eredményeit egy későbbi alfejezetben mutatom be.

31. táblázat - Többcélú, többprojektes futtatás egyedi célfüggvény értékek szerinti kiértékelése

Teszt száma	Célfüggvény prioritása						Célfüggvény értéke					
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
1	1	0	0	0	0	0	-8	0	0	0	321	79707
2	0	1	0	0	0	0	0	0	0	0	302	79557
3	0	0	1	0	0	0	0	0	0	0	311	80470
4	0	0	0	1	0	0	256	256	436	2	323	84605
5	0	0	0	0	1	0	237	237	637	4	276	79837
6	0	0	0	0	0	1	227	227	548	4	289	72393

6.3.3 Több célfüggvény együttes vizsgálata

A második tesztcsoportban már több célfüggvény együttes használatát vizsgáltam. Az ütemezési feladat az első tesztcsoporthoz képest nem változott. A célfüggvényeket három szempontrendszer szerint csoportosítottam és a csoportoknak megfelelően teszteltem a különböző fontossági prioritások értékeit. A figyelembe vett csoportok:

1) Késések minimalizálása ($L_{max}, T_{max}, T_{sum}, U_{sum}$),
2) Utolsó feladat befejezési időpontjának minimalizálása (C_{max}),
3) Feladatok összegzett befejezési idejének minimalizálása (C_{sum}).

A 7. teszt részletes eredményeit a 11.2 függelék tartalmazza. Több célfüggvény együttes vizsgálata miatt az algoritmus nagyobb gyakorisággal talál javítást valamelyik célfüggvény tekintetében. Ennek következménye, hogy így a javító lépések táblázata nagyobb méretű. Terjedelmi okok miatt minden teszt részletes futási eredményeit a disszertációban így nem áll módomban közölni, ezért az eredmények elemezhetősége miatt a további tesztek esetén a függelék csak a végrehajtott tesztek eredményeiből készített karakterisztika diagrammokat tartalmazza. Az 32. táblázat tekinti át az elvégzett tesztek esetén megadott célfüggvény prioritásokat és a keresés által elért eredményeket. A teszt során manuálisan változtattam a célfüggvények prioritásán és vizsgáltam a beállítások módosításának hatását a célfüggvények eredményeire. A továbbiakban a szisztematikusan elvégzett teljesítménytesztek egy illusztratív kivonatát mutatom be és értékelem az eredményeket.

32. táblázat - Kombinált célfüggvény értékek szerint végrehajtott tesztek eredménye

Teszt száma	**Célfüggvény prioritása**						**Célfüggvény értéke**					
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
7	1	1	1	1	1	1	-7	0	0	0	287	73837
8	0	10	20	8	1	1	0	0	0	0	284	74760
9	0	2	2	2	1	0	0	0	0	0	283	79365
10	5	0	0	0	1	0	-7	0	0	0	280	79193
11	0	0	2	0	0	1	-2	0	0	0	297	73819

A 7. és 8. teszt során mindhárom szempontrendszert együttesen vettem figyelembe. A két teszt célja, hogy a határidő betartása mellett mennyire közelíthető meg a C_{sum} és C_{max} korábbi egyedüli minimuma. A 7. teszt indítási paraméterei inkább a C_{sum}-nak, míg a 8. teszt inkább a C_{max}-nak kedvezett.

A 9. és 10. tesztben a C_{sum} célfüggvényt nem vettem figyelembe, ezért nulla prioritást állítottam be. Megfigyelhető, hogy ez a C_{max} számolt eredményeinek kedvezett (előbb 287 és 284, majd 283 és 280). A 10. teszt sorozat C_{max} szempontjából legjobb értékének az L_{max} célfüggvénynek volt szerepe, mivel annak segítségével nem csak nulla késésig tolta előre a

határidős projekteket, hanem még sietést is tudott generálni. A generált sietés pedig a C_{max} javítását eredményezte.

A 11. tesztben a határidő betartása mellett csak a C_{sum} volt figyelembe véve. Itt a T_{sum} egyedül is tartani tudta a nulla határidő-túllépést. A C_{sum} értéke egészen jól lecsökkent. Ennél jobb értéket a korábbi tesztekben csak az önállóan használt C_{sum} célfüggvény esetén sikerült elérni (6. teszt).

Megfigyelés: Ebben a többprojektes feladatban az U_{sum} célfüggvény továbbra sem teljesített jól. Lehetséges indokként merült fel, hogy a célfüggvény értékkészlete kicsi ([0,5]) és a keresés során a célfüggvény javulásából származtatható javítási információ csekély.

6.3.4 Projekt határidő alternatív leképzésének vizsgálata

A második tesztcsoport sejtésének igazolására vagy cáfolására további tesztek kerültek végrehajtásra. A többprojektes feladatot olyan módon módosítottam, hogy a határidővel rendelkező projektek esetén a projekthez tartozó minden feladat számára a projekt teljesítési határideje került teljesítési határidőként beállításra. Ez a leképzés helyességén nem változtat, hiszen a projekt teljesítési határideje, ha egyébként feladat szintű teljesítési határidőket nem határoztunk meg, úgy a projekt minden feladata tekintetében maximális teljesítési határidőként tekinthető.

A 12-17. számú tesztek célfüggvény prioritásait és a célfüggvények eredményeit a 33. táblázat ismerteti.

33. táblázat - Projekt határidejét a projekt feladataira képező modell teljesítményvizsgálata

Teszt száma	**Célfüggvény prioritása**						**Célfüggvény értéke**					
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
12	0	0	0	1	0	0	-2	0	0	0	304	79814
13	1	0	0	0	0	0	-8	0	0	0	321	79707
14	0	1	0	0	0	0	0	0	0	0	302	79557
15	0	0	1	0	0	0	-1	0	0	0	331	80912
16	0	0	10	0	1	0	213	0	0	0	279	79377
17	0	0	10	0	0	1	0	0	0	0	292	73797

A 12. teszt igazolta a korábbi feltételezés helyességét. **Ugyanannak a problémának egy másik megfogalmazása ugyanazzal a modellel és ütemezési módszerrel jobb megoldáshoz vezet**, mivel a kereső a több feladatnál is feltüntetett határidő túllépések visszajelzései alapján hatékonyabban tud javító lépést tenni. Az eredmények további ellenőrzése érdekében a 12. tesztet még további négy alkalommal megismételtem, minden esetben más véletlenszerűen választott kiindulási megoldásból indulva. Az ismételt tesztfuttatások esetén a következő eredményeket kaptam.

34. táblázat - U_{sum} célfüggvény teljesítménymérése különböző keresés indítási feltételek mellett

Teszt száma	Célfüggvény prioritása						Célfüggvény értéke					
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
12	0	0	0	1	0	0	-2	0	0	0	304	79814
12 _ 2	0	0	0	1	0	0	-8	0	0	0	324	83253
12 _ 3	0	0	0	1	0	0	-1	0	0	0	313	80991
12 _ 4	0	0	0	1	0	0	-3	0	0	0	301	80677
12 _ 5	0	0	0	1	0	0	-2	0	0	0	322	80118

A 34. táblázatban ismertetett eredmények alapján látható, hogy a kereső az új leképzést követően mind az 5 ellenőrző esetben megtalálta az U_{sum} = 0 optimumot. Az U_{sum} szerinti optimalizálás hatékonyság növekedése után ellenőriztem, hogy az új leképzés milyen további hatással van a már korábban végrehajtott tesztek eredményére.

A 13. teszt eredménye teljes mértékben megegyezik az 1. teszt eredményével. Ez várható volt, hiszen L_{max} optimalizálás szempontjából a feladatok egyedi határideje nem bír jelentőséggel.

A 14. teszt eredménye teljes mértékben megegyezik a 2. teszt eredményével, mivel T_{max} szempontjából a feladatok egyedi határideje releváns.

A 15. teszt eredményét a 3. teszt eredményével hasonlítottam össze. T_{sum} szempontjából megtalálja a referencia értéket, azonban azt figyeltem meg, hogy ehhez kevesebb lépésszámra van szüksége. Ennek kiértékelését a 36. táblázatban tüntetem fel. Minden esetben azt tapasztaltam, hogy a szükséges iterációk száma az új feladat definiálás esetén csökkent.

35. táblázat - Az első tesztcsoport és a harmadik tesztcsoport eredményeinek összehasonlító vizsgálata L_{max},T_{max} és T_{sum} esetén

Teszt száma	Célfüggvény prioritása						Célfüggvény értéke					
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
1	1	0	0	0	0	0	-8	0	0	0	321	79707
13	1	0	0	0	0	0	-8	0	0	0	321	79707
2	0	1	0	0	0	0	0	0	0	0	302	79557
14	0	1	0	0	0	0	0	0	0	0	302	79557
3	0	0	1	0	0	0	0	0	0	0	311	80470
15	0	0	1	0	0	0	-1	0	0	0	311	81878

36. táblázat - Legjobb T_{sum} érték megtalálásához szükséges iterációk számának összehasonlítása

Teszt száma	Célfüggvény prioritása						Célfüggvény értéke						Iteráció
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}	
3	0	0	1	0	0	0	0	0	0	0	311	80470	2863
15	0	0	1	0	0	0	-1	0	0	0	311	81878	1050
15_2	0	0	1	0	0	0	0	0	0	0	315	80912	1751
15_3	0	0	1	0	0	0	-1	0	0	0	318	82108	1289
15_4	0	0	1	0	0	0	-2	0	0	0	307	79831	1249
15_5	0	0	1	0	0	0	-4	0	0	0	315	80735	1074

A 16. tesztben sikerült a határidők betartása mellett tovább csökkenteni egy egységgel a C_{max} értékét. Az első tesztsorozatban 280 volt a legjobb érték, míg itt 279-et ért el a módszer.

A 17. tesztben sikerült a határidők betartása mellett a C_{sum} értékét csökkenteni. Az első tesztsorban a legjobb érték 73819 volt, míg ez 73797 értékre csökkent.

Az elért eredmények igazolják, hogy a feladat megfogalmazása nagymértékben befolyásolja a megoldás minőségét, ezért indokolt kifinomult optimalizálási modelleket alkalmazni. Az értekezés első tézise éppen egy ilyen finomhangolható, rugalmas modellt fogalmaz meg.

6.3.5 Késedelmes projektek vizsgálata

Ebben a tesztcsoportban azt vizsgáltam, hogy az MOSM módszer hogyan kezeli azt az esetet, amikor biztosan lesz késés. Ennek vizsgálatához a negyedik tesztcsoportban a feladatok határidejét a második tesztcsoporthoz képest módosítottam. Az öt határidős projekt minden taszkjának egyetlen közös határidőt állítottam be. A beállított határidő 39 időegység. Ez az érték megegyezik a korábbi tesztekben használt legkisebb határidővel.

Ha nem végezhető el minden taszk határidőre, akkor a T_{max}, T_{sum}, U_{sum} célfüggvények is egymás konkurenseivé válhatnak vagy akár egymás hatását erősíthetik is. Ilyen esetben feltehetőek a következő kérdések: Ha elkerülhetetlen a határidő-túllépés, akkor több kis csúszás, vagy inkább kevesebb nagy csúszás lenne elfogadhatóbb? Esetleg a csúszások összegét szeretnénk célszerűen minimalizálni?

Ebben a tesztcsoportban 8 tesztsorozatot végeztem el, tesztsorozatonként 5 tesztet futtattam azonos beállítással. A 37. táblázatban látható a célfüggvények beállított prioritásai és elért értékei. A táblázat a tesztsorozaton belül a legjobb teszt értékeit mutatja. A 18 - 21. tesztsorozat során egyedi célok szerint kerestem optimumot, rendre L_{max}, T_{max}, T_{sum} és U_{sum} szerint.

37. táblázat - Biztosan késedelmes projektek vizsgálata különböző célfüggvény prioritások szerint

Teszt sorozat	Célfüggvény prioritása						Célfüggvény értéke					
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
18	1	0	0	0	0	0	25	25	553	38	309	80690
19	0	1	0	0	0	0	25	25	553	38	309	80690
20	0	0	1	0	0	0	33	33	492	35	344	82231
21	0	0	0	1	0	0	258	258	2848	28	325	81729
22	0	1	1	1	0	0	30	30	555	37	313	78621
23	0	1	1	10	0	0	35	35	575	29	313	81613
24	0	10	1	1	0	0	24	24	502	38	327	82411
25	0	1	10	1	0	0	29	29	459	31	321	80045

A 18. és 19. teszt eredménye teljesen megegyezik egymással. Ez várható volt hiszen, ha biztosan van csúszás, akkor az L_{max} értéke nagyobb, mint nulla és megegyezik a T_{max} értékével.

A 19., 20., és 21. tesztekben a T_{max}, T_{sum} és U_{sum} külön-külön a saját szempontjából jobb eredményt adott, mint a másik kettőnél elért megoldások. A rendszerben közösen használva ezeket a célfüggvényeket kompromisszumos megoldások állíthatók elő. Erre mutat példát a 22. teszt.

A 23., 24., és 25. tesztben a három vizsgált célfüggvény prioritását úgy állítottam be, hogy egyet kiemeltem 10 prioritásértékkel és a másik kettőt egységnyi prioritásértékkel használtam. Az eredményekből látható, hogy a nagyobb prioritás hatással volt a keresésre és ez az eredményekben is megjelenik.

A 24. tesztsorozatban alkalmazott prioritásértékek kombinációja kiváló eredményeket adott. Az 5 ismételt végrehajtás eredményeit mutatja be a 38. táblázat.

38. táblázat - Biztosan késedelmes projektek ismétlő vizsgálata meghatározott célfüggvény prioritások szerint

Teszt sorozat	Célfüggvény prioritása						Célfüggvény értéke					
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
24	0	10	1	1	0	0	26	26	594	41	317	81451
24_2	0	10	1	1	0	0	29	29	481	33	309	79879
24_3	0	10	1	1	0	0	25	25	464	35	318	80228
24_4	0	10	1	1	0	0	28	28	506	33	307	78702
24_5	0	10	1	1	0	0	24	24	502	38	327	82411

Megfigyelhető, hogy a 24_2 futtatás során T_{sum} számított értéke 481, ami jobb, mint a 20. tesztben elért eredmény, ahol mindössze a T_{sum} célfüggvényt vette figyelembe a kereső és 492 értéket kaptam. A 24_3 esetén T_{sum} értéke 464, miközben T_{max} értéke 25. A 24_5 futtatás

során T_{max} értéke 24, ami az eddigi legjobb T_{max} érték. Ez a T_{max} jobb, mint az önmagában használt T_{max} célfüggvény a 19. tesztben.

A 24. tesztsorozathoz hasonló módon az eredmények kiértékelésekor a 25. tesztsorozatban is 5 futtatást végeztem. Az elért eredmények a 39. táblázatban láthatók.

39. táblázat - Biztosan késedelmes projektek ismétlő vizsgálata meghatározott célfüggvény prioritások szerint

Teszt sorozat	Célfüggvény prioritása						Célfüggvény értéke					
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
25	0	1	10	1	0	0	29	29	522	39	317	80272
25_2	0	1	10	1	0	0	31	31	493	33	322	81748
25_3	0	1	10	1	0	0	36	36	519	32	308	81908
25_4	0	1	10	1	0	0	29	29	459	31	321	80045
25_5	0	1	10	1	0	0	35	35	563	35	324	81841

A 25. tesztsorozatban beállított prioritásértékek együtt kiemelkedő eredményt értek el. A 25_4 teszt esetén a T_{sum} értéke 459, ami a legjobb megtalált T_{sum} érték a tesztcsoport esetén.

A 24. és 25. tesztsorozatokból az a fontos következtetés vonható le, hogy a megfelelően megválasztott prioritásértékek adott esetben jobb eredményt is elérhetnek, mint a külön-külön alkalmazott célfüggvények által elérhető értékek.

6.4 Több feladatválasztási szabály együttes vizsgálata

A korábbi tesztek esetén minden esetben az ütemezést felépítő eljárás egyetlen feladatválasztási szabályt, a kereső modul által meghatározott CCV értékeket figyelembe vevő feladatválasztási szabályt alkalmazta.

Ebben a fejezetben ismertetett tesztsorozatban kifejezetten az került megvizsgálásra, hogy milyen futtatási eredményeket képes az ütemező módszer abban az esetben biztosítani, amikor a kereső modult nem használjuk. Ez a futtatási mód alkalmas arra, hogy az ütemezőt reaktív módon lehessen olyan környezetben alkalmazni, amikor a kereső eljárások végrehajtására nincs elegendő keretidő, vagy számítási kapacitás. A teszt további célja több feladatválasztási szabály együttes alkalmazásának összehasonlítása volt az egyedi feladatválasztási szabályokhoz képest.

A tesztek a PSLIB benchmark feladatokon kerültek végrehajtásra. A PSLIB benchmark feladat és annak leképzése az RCMOMPSP modellre a 6.2 fejezetben bemutatott módon történt. A benchmark feladatokat manuálisan előre kiválasztott feladatválasztási szabályokon és előre meghatározott prioritások és optimalizálási irányok alkalmazásával teszteltem.

Az alkalmazott feladatválasztási szabályokat mutatja be a 40. táblázat.

40. táblázat - Feladatválasztási szabályok

Szabály rövidítése	Szabály neve	Szabály definíciója
Nsucc	Követő feladatok száma (*Number of successors*)	Annak a feladatnak nagyobb/kisebb a prioritása, amely feladatnak több követő feladata van.
ProcT	Végrehajtási idő (*Processing time*)	Annak a feladatnak nagyobb/kisebb a prioritása, amely feladatnak nagyobb a végrehajtási ideje.
CPM_EST	CPM ütemezés szerinti legkorábbi indítási időpont (*CPM Earliest Start Time*)	Annak a feladatnak nagyobb/kisebb a prioritása, amely feladatnak hamarabb/később van a CPM módszer szerinti legkorábbi indítási időpontja.
CPM_EFT	CPM ütemezés szerinti legkorábbi befejezési időpont (*CPM Earliest Finish Time*)	Annak a feladatnak nagyobb/kisebb a prioritása, amely feladatnak hamarabb/később van a CPM módszer szerinti legkorábbi teljesítési időpontja.
CPM_LST	CPM ütemezés szerinti legkésőbbi indítási időpont (*CPM Latest Start Time*)	Annak a feladatnak nagyobb/kisebb a prioritása, amely feladatnak hamarabb/később van a CPM módszer szerinti legkésőbbi indítási időpontja.
CPM_LFT	CPM ütemezés szerinti legkésőbbi befejezési időpont (*CPM Latest Finish Time*)	Annak a feladatnak nagyobb/kisebb a prioritása, amely feladatnak hamarabb/később van a CPM módszer szerinti legkésőbbi befejezési időpontja.
RC_DEST	Erőforráskorlátokat figyelembe vevő dinamikus legkorábbi indítási időpont (*Resource constraint dynamic earliest start time*)	Annak a feladatnak nagyobb/kisebb a prioritása, amely feladat esetén az aktuálisan elkészített rész-ütemterv szerint a legnagyobb/legkisebb a dinamikusan számolt legkorábbi indítási időpontja
DD	Feladat teljesítési határideje (Due Date)	Annak a feladatnak nagyobb/kisebb a prioritása, amely feladatnak hamarabb/később van a teljesítési határideje.

A PSLIB J30 benchmark feladat nem tartalmaz határidőket, ezért a DD feladatválasztási szabály alkalmazása a negatív teszt miatt került be a tesztelt szabályok közé. A megadott nem nulla prioritásnak nem szabad, hogy a végeredményt befolyásoló szerepe legyen. A teszt során minden benchmark feladat esetén egyetlen teszt került végrehajtásra, mivel az algoritmus működése determinisztikus (a számítás során nem kerül véletlenszám generálásra).

A korábbi mérésekhez hasonlóan előbb minden egyes feladatválasztási szabályt egyedileg vizsgáltam meg. Ekkor csak egy feladatválasztási szabály prioritása volt nem nulla. A kapott eredményeket referenciaértékként használom a több feladatválasztási szabályt alkalmazó futási eredmények tekintetében.

A 41. táblázatban láthatóak az egyedi feladatválasztási szabályok által elért eredmények. Látható, hogy az Nsucc feladatválasztási szabály esetén a minimalizálás és a maximalizálás is megvizsgálásra került.

41. táblázat - PSLIB 30 benchmark feladatok futási eredményei egyedi feladatválasztási szabály alkalmazásával

Feladatválasztási szabály prioritiása és az optimálási irány (minimalizálás: -1, maximalizálás: 1) szorzata								Teljesítmény mutató	
RC_DEST	Nsucc	ProcT	CPM_EST	CPM_EFT	CPM_LST	CPM_LFT	DD	Optimális találatok száma (max 480)	Átlagos relatív eltérés (ADR)
1	0	0	0	0	0	0	0	157	0,07826
0	1	0	0	0	0	0	0	142	0,140896701
0	-1	0	0	0	0	0	0	160	0,1009952989
0	0	1	0	0	0	0	0	136	0,1741794254
0	0	0	1	0	0	0	0	148	0,09147926868
0	0	0	0	1	0	0	0	134	0,1154640351
0	0	0	0	0	1	0	0	245	0,04922669189
0	0	0	0	0	0	1	0	243	0,05119640813
0	0	0	0	0	0	0	1	154	0,09535507438

Az egyedi méréseket követően több feladatválasztási szabályt alkalmaztam egyidejűleg. A 42. táblázatban láthatóak a különböző beállítások és a tesztek eredményei. A táblázatok alapján megállapítható, hogy átlagos relatív eltérés tekintetében a feladatválasztási szabályok kombinálása a legtöbb esetben jobb eredményt ad, mint amit az egyedi szabályok önmagukban el tudtak érni.

42. táblázat - PSLIB 30 benchmark feladatok futási eredményei több feladatválasztási szabály együttes alkalmazásával

Feladatválasztási szabály prioritása és az optimalizálási irány (minimalizálás: -1, maximalizálás: 1) szorzata								Teljesítmény mutató	
RC_DEST	Nsucc	ProcT	CPM_EST	CPM_EFT	CPM_LST	CPM_LFT	DD	Optimális találatok száma (max 480)	Átlagos relatív eltérés (ADR)
1	0	0	0	0	2	0	0	250	0,04557126916
1	-1	-1	1	1	1	1	0	210	0,06159341688
1	1	1	1	1	1	1	1	170	0,09205371544
1	1	-1	1	1	1	1	1	198	0,0725273943
1	-1	1	1	1	1	1	1	184	0,07718735328
1	-1	-1	1	1	1	1	1	210	0,06159341688
1	-1	-1	1	1	10	1	1	247	0,0514606567
1	-1	-1	1	1	20	1	1	247	0,04951052151
1	-1	-1	1	1	5	1	1	241	0,05267209916
2	0	0	0	0	1	0	0	244	0,03840760371
5	0	0	0	0	4	0	0	247	0,04088550713
5	-1	-1	1	1	1	1	1	206	0,05537005827
6	0	0	0	0	3	0	0	244	0,03840760371
6	-0,1	0,1	0	0	3	0	0	248	0,03775955726
6	-0,5	0,5	0	0	3	0	0	243	0,0372489305
6	-0,5	0,5	0	0	3	0,2	0	243	0,03832872747
6	-0,5	0,5	0	0	3	-0,2	0	245	0,0374791767
6	-0,5	0,5	0	0	3	0,5	0	247	0,03744202742
6	-0,5	0,5	0	0	3	1	0	250	0,03784784792
6	-0,5	0,5	0	0	3	-1	0	237	0,0386707721
6	-0,5	0,5	0	0,1	3	0	0	241	0,03755601349
6	-0,5	0,5	0	-0,1	3	0	0	246	0,03732367137
6	-0,5	0,5	0	0,5	3	0	0	233	0,03777206395
6	-0,5	0,5	0,05	0,05	3	0,05	0,05	243	0,03763458738
6	-0,5	0,5	0,1	0	3	0	0	243	0,03784250049
6	-0,5	0,5	-0,1	0	3	0	0	242	0,03799934644
6	-0,5	0,5	0,1	0,1	3	0,1	0,1	242	0,03812645366
6	-0,5	0,5	0,5	0	3	0	0	235	0,03937497544
6	-0,5	0,5	-0,5	0	3	0	0	245	0,04105560508
6	-0,5	0,5	1	0	3	0	0	230	0,04114086199
6	1	1	0	0	3	0	0	230	0,0488259021
6	-1	1	0	0	3	0	0	231	0,04354913627
6	-1	-1	0	0	3	0	0	228	0,05638278338
6	-1	-1	1	1	3	1	1	222	0,05000295283

A mérés során kiemeltem az ARD szempontjából itt mért legjobb eredményt: 0,0372489305.

A legjobb eredmény kiemelése mellett fontos megvizsgálni az eljárás hatékonyságát, ha több mint egy feladatválasztási szabályt alkalmazunk. A 41. és 42. táblázatban ismertetett futási eredmények alapján a harmadik legjobb eredményt a RC_DEST feladatválasztási szabály egyedüli alkalmazása szolgáltatta, ahol az ARD értéke körülbelül 0,07826 volt (41. táblázat, 1. sor). Amikor ezt a feladatválasztási szabályt egy másik szabállyal együttesen alkalmaztam, akkor még jobb eredményeket értem el. Például, a RC_DEST feladatválasztási szabály 1 prioritás értékével, és a CPM_LST feladatválasztási szabály 2 prioritás értékével (42. táblázat, 1. sor), az ARD értéke körülbelül 0,04557 volt. A két kiválasztási szempont prioritásának cseréje (42. táblázat, 10. sor) csökkentette az ARD-t körülbelül 0,03841-re. Érdekes megfigyelés volt, hogy a kombinált megoldásban előnyösebb volt nagyobb prioritást adni annak a feladatválasztási szempontnak, amelyet csupán egyedül alkalmazva rosszabbul teljesít, a másik feladatválasztási szempont, amely egyedül használva jobb eredményt ér el. Megfigyelhető továbbá, hogy az feladatválasztási szempontok során adott prioritások konkrét értéke nem volt olyan fontos, mint azok relatív aránya. Például két feladatválasztási szempont prioritás értékének a megháromszorozása, vagyis 6:3-ra állítása 2:1 helyett, nem változtatta meg az ARD értékét.

Ez a kísérleti eredmény matematikailag is bizonyítható. Ha figyelembe vesszük az *F* függvényt, amely a relatív összehasonlítás alapját képezi, akkor világossá válik, hogy a prioritás értékek egy adott állandóval való megszorzása ekvivalens az *F* függvény végleges eredményének ugyanezzel az állandóval való megszorzásával, amelyet az összeadásból emelhetünk ki. A definíció szerint az *F* függvény eredményét összehasonlítjuk a nullával, ami meghatározza az összehasonlítás eredményét. Ezért, ha az *F* függvény eredményét bármilyen nem-negatív valós számmal megszorozzuk, az nem változtatja meg a nullához való viszonyát. Ha az *F* függvény értéke nagyobb volt nullánál, akkor nagyobb marad; ha kisebb volt nullánál, akkor kisebb is marad; és ha nulla volt, akkor nulla is marad. Ezt a tényt figyelembe kell venni a feladatválasztási szabályok prioritásértékeinek finomhangolásakor.

A további feladatválasztási szempontok alkalmazása tovább javította az eredményeket. A 42. táblázat példái azt mutatják, hogy az ARD érték jelentősen csökkenthető különböző prioritási értékek alkalmazásával. A RC_DEST és CPM_LST szempontok továbbra is uralkodó szerepet játszottak, de más szempontok is hasznosnak bizonyultak kisebb prioritásértékek mellett. Kiemelendő, hogy a NSucc és ProcT használata is hasznos volt. Ezek a feladatválasztási szempontok befolyásolják a döntést olyan esetekben, amikor az *F* függvény értéke közel állna a nullához a nagyobb prioritású szempontok alapján. A NSucc prioritása negatív, mert e szempont esetében a módszer előnyben részesíti a jellemző számértéket, ha az nagyobb. Az Nsucc – követő feladatok száma – esetében ezért a negatív prioritás használata előnyös. A többi szempont esetében a kisebb szám kedvezőbb.

Ebben a tesztben a legjobb eredményt az alábbi feladatválasztási prioritások adták: 6; -0.5; 0.5; 0; 0; 3; 0; 0 (42. táblázat, 15. sor). Az így elért ARD értéke körülbelül 0,037249 volt. A 42. táblázat alapján látható, hogy 0,038 alatti ARD érték elérhető számos más beállítással is. Ez az eljárás kedvező tulajdonsága, mert egy adott specifikus rendszer esetén a beállított

prioritási értékrendszer hatékonyan használható, és elegendően jó megoldást biztosít, miközben a rugalmasságát is megőrzi.

Az eredmények összefoglaló kiértékelése

A mérés során kézzel állítottam be az alkalmazott feladatkiválasztási szempontok prioritásait, és megfigyeltem, hogy a döntési szempontok mely kombinációja nyújtott jobb teljesítményt, mint bármelyik szempont önmagában. Ez az eredmény megerősítette azt a hipotézisemet, hogy előnyös a döntési szempontokat a javasolt relatív minősítési modellel kombinálva alkalmazni.

A több feladatválasztási szabályt egyszerre, súlyozottan és optimalizálási iránynak megfelelően érvényesítő kiterjesztés képes alkalmazkodni az ütemezési problémák széles köréhez, mivel az új relatív összehasonlítás természete problémafüggetlen, és könnyen be tud vonni különböző döntési szempontokat. Bármilyen új feladatkiválasztási szempont hasonló módon bevezethető. A felhasználó hozzárendelheti a döntési szempontok számszerű értékének kiszámítását az egyes feladatokhoz, kalibrálhatja az egyes döntési szempontok egyéni prioritását, és a módszer máris figyelembe veszi az új kritériumokat. A valódi tudáselemek felfedezhetők a döntési szabályok halmazában és a hozzárendelt prioritások értékeiben.

Az alkalmazott módszer így rugalmasan alkalmazkodik a változó optimalizálási célokhoz. A javasolt relatív összehasonlításon alapuló több szempontú döntési modell hatékonyan és gyorsan képes megoldani más kiválasztási vagy optimalizálási problémákat. Tekintettel arra, hogy a javasolt több szempontú kiválasztási modell kizárólag problémafüggetlen elemeket tartalmaz, így bármilyen típusú döntéshozatali probléma megoldási folyamatában támogató komponensként alkalmazható.

Fontos megjegyezni, hogy a mérés során vizsgált ARD értékek csökkenthetők keresési algoritmusok alkalmazásával, azonban ezek jelentősen hosszabb számítási idővel járnak. Egy reaktív konstrukciós algoritmus csak egy megoldást hoz létre, míg a keresési algoritmusoknak iteratívan kell generálniuk egy viszonylag nagy számú megoldást, ami sokkal hosszabb futási időt eredményez.

A tesztelés során kézzel kalibráltam az eljárást, de további módszerek is alkalmazhatók a prioritások beállítására. Például keresési algoritmusok használhatók a prioritások finomhangolására a kívánt alkalmazási környezet szerint. Gépi tanulási technikák is bevethetők a prioritási értékek meghatározására. Ezek a fejlesztési irányok, többek között, a jövőbeli kutatási feladatok alapját képezik.

7 Új tudományos eredmények

1. Tézis: Optimalizálási modell többcélú, többprojektes, erőforrás-korlátos ütemezési feladatok megoldásának támogatására.

Kidolgoztam egy optimalizálási modellt független és részben összefüggő projektek aktivitásainak (taszkjainak) ütemezésére, amely magába foglalja az ismert erőforráskorlátos projektütemezési (RCPSP) problémán túlmenően a megújuló erőforrások időben változó kapacitás-korlátait, az aktivitások és a projektek saját célfüggvényeit, a teljes rendszerre vonatkozó célfüggvényeket, valamint a célfüggvények egyedi prioritásait és optimalizálási irányait. A kidolgozott matematikai modell definiálja ennek a kiterjesztett MORCMPSP problémának a döntési változóit, a korlátfeltételeit és a rugalmasan változtatható célfüggvény-rendszerét.

Az optimalizálási modell fontosabb jellemzői:

1.1. A rugalmas célfüggvény-rendszer bevezetése lehetővé teszi tetszőleges számú, dimenziójú, értékkészletű és optimalizálási irányú teljesítménymutató figyelembevételét az ütemezés során.

1.2. A klasszikus RCPSP modellben az erőforrás kapacitáskorlátja csupán egy időfüggetlen konstans érték lehet, míg az én RCMOMPSP modellem tetszőleges kapacitás-idő függvények használatát biztosítja.

1.3. A projektek lehetnek egymástól függetlenek, vagy lehet közös aktivitásuk. Adott aktivitás egyidejűleg több projekthez is tartozhat.

1.4. Jól ismert ütemezési feladatok megoldásán keresztül igazoltam, hogy az új modell alkalmas a klasszikus (single mode RCPSP) projektütemezési feladaton túlmenően diszkrét gyártási folyamatok ütemezési feladatainak kezelésére is.

Az 1. tézisben összefoglalt eredményeket az értekezés 3. fejezetében részletesen ismertettem, valamint a [S19] publikációban is bemutattam.

2. Tézis: Hibrid ütemezési módszer kiterjesztett többcélú, többprojektes, erőforráskorlátos ütemezési feladatok megoldására.

Egy kombinált hibrid megoldási módszert dolgoztam ki kiterjesztett többcélú, többprojektes, erőforráskorlátos ütemezési feladatok megoldására, amely a probléma megoldását több döntési szintre bontja. Az első szinten kereső algoritmusok állítják be az aktivitások (taszkok) prioritását, majd a második szinten a taszkok prioritásai által vezérelt felépítő algoritmusok készítenek megvalósítható megoldásokat. A jelölt megoldások közötti választást többcélfüggvényes relatív minősítésen alapuló modell támogatja.

A módszer fontosabb jellemzői:

2.1. A kidolgozott hibrid módszer a szándékos várakoztatást nem igénylő ütemezési feladatok megoldásának támogatására készült. A sietést minimalizáló célfüggvényeket leszámítva tetszőleges célfüggvények alkalmazhatók a megoldások minőségének számszerűsítésére.

2.2. A módszer integráltan együtt kezeli az erőforrásokra, a projektekre és az aktivitásokra (taszkokra) vonatkozó korlátozásokat.

2.3. A módszer figyelembe veszi a projektek és a taszkok RCMOMPSP modellben meghatározott jellemzőit.

2.4. A hibrid módszer rugalmas alkalmazhatóságát és hatékonyságát az implementált szoftverrel megoldott tesztfeladatok eredményei igazolták.

A 2. tézisben összefoglalt eredményeket az értekezés 4. és 6. fejezeteiben részletesen ismertettem, valamint az [S19, S18, S17, S16, S15] publikációkban is bemutattam.

3. Tézis: Többszempontú taszkválasztó módszer felépítő jellegű ütemezési módszerek számára.

Kidolgoztam egy többszempontú taszkválasztó módszert az ütemezési feladatok megoldásait felépítő módszerek rugalmasságának növelésére, amely a relatív változás-orientált összehasonlítási modellt felhasználva lehetővé teszi sok kiválasztási szempont priorizált együttes figyelembevételét azokban az esetekben is, amikor a kiválasztási szempontokat reprezentáló számértékek vegyes előjelűek és eltérő optimalizálási irányt támogatnak.

A módszer fontosabb jellemzői:

3.1 A felépítő jellegű ütemező algoritmusok egyik legfontosabb döntési fázisa a következő beütemezendő aktivitás (taszk) kiválasztása az ütemezhető taszkok halmazából. A kidolgozott új módszer lehetővé teszi, hogy egynél több kiválasztási szempontot vegyünk egyidejűleg figyelembe.

3.2 A kiválasztási szempontok között egyaránt szerepelhetnek a klasszikus taszk-orientált prioritás-elvű számértékek, dinamikusan újra számolt időfüggő számértékek és külső irányítási hatást kifejező vezérlő számértékek.

3.3 A döntési szempontokat kifejező számértékek egyaránt lehetnek pozitív és negatív előjelűek, valamint kisebb vagy nagyobb számérték egyaránt képviselheti az adott szempont döntésre gyakorolt hatását.

3.4 Minden egyes döntési szemponthoz egy saját prioritási érték is tartozik, amely az adott szempont figyelembevételének mértékét szabályozza.

3.5 Ezek a kiterjesztések együtt lehetővé teszik a kiválasztási módszer felhasználó általi kalibrálását.

A 3. tézisben összefoglalt eredményeket az értekezés 4.2 alfejezetében részletesen ismertettem, valamint az [S19, S18] publikációkban is bemutattam.

4. Tézis: Kombinált keresési metaheurisztika permutációs sorrendi feladatok megoldására.

Populáció-alapú evolúciós és lokális szomszédsági stratégiákat együttesen alkalmazó metaheurisztikus keresési módszert dolgoztam ki permutációs sorrendi feladatok megoldására (MOSM). A módszer hatékonyan alkalmazható az aktivitások (taszkok) prediktív ütemezési és végrehajtási sorrendjének meghatározására.

A módszer fontosabb jellemzői:

4.1 A kidolgozott kombinált keresési módszer előnyösen alkalmazható prediktív ütemezési feladatok megoldására, amikor megfelelő idő áll rendelkezésre sok megoldás előállítására és kiértékelésére.

4.2 A keresési módszer megoldást módosító operációinak működési módja független a célfüggvényektől.

4.3 A módszer hatékonyan kezeli azokat a feladatokat is, amelyekben az aktivitásoknak (taszkoknak) megelőzési előfeltételei vannak. A beépített normalizáló algoritmus biztosítja, hogy a sorrend minden korlátozásnak megfelel.

4.4 A módszer hatékonyságát a benchmark tesztfeladatokon elért eredmények igazolták.

A 4. tézisben összefoglalt eredményeket az értekezés 4.4.1 és 4.5 fejezeteiben részletesen ismertettem, valamint az [S18] publikációban is bemutattam.

5. Tézis: Integrált rekombinációs és mutációs operátor a JAYA keresési metaheurisztika hatékonyságának növelésére.

Többszülős rekombinációt és mutációt egyidejűleg megvalósító operátort dolgoztam ki (38) a JAYA keresési metaheurisztika hatékonyságának növelésére. Az új operátor probléma-független, így alkalmas tetszőleges optimalizálási feladat megoldásának támogatására. Az operátor hatékonyabban működik az $1||C_{sum}$ benchmark feladatokon mint az eredeti JAYA operátor.

A módszer fontosabb jellemzői:

5.1 A kidolgozott új operátor egyidejűleg valósít meg többszülős rekombinációt és mutációt.

5.2 Az új operátor működési módja független a célfüggvényektől és a korlátozásoktól.

5.3 Az új operátor hatékonyságának tesztelését $1||C_{sum}$ ütemezési feladatokon végeztem el úgy, hogy a JAYA keresési elvet alkalmaztam az elemi ütemezendő aktivitások (taszkok) prioritásainak beállítására. A kiosztott prioritások alapján a soros generálási sémát használtam a megoldás előállítására. A benchmark tesztfeladatokon elért eredmények igazolták, hogy az új operátor kevesebb keresési lépésszámmal oldja meg a feladatokat, mint az eredeti JAYA operátor.

Az 5. tézisben összefoglalt eredményeket az értekezés 4.5.2 és 6.1.2 fejezeteiben részletesen ismertettem, valamint az [S16], publikációkban is bemutattam.

8 New Scientific Results

Thesis 1: Optimization model to support the solution of multi-objective, multi-project, resource-constrained scheduling problems.

I developed an optimization model for scheduling the activities (tasks) of independent and partially interrelated projects. This model includes, in addition to the well-known Resource-Constrained Project Scheduling Problem (RCPSP), the time-varying capacity constraints of renewable resources, the individual objective functions of activities and projects, the overall system-wide objective functions, as well as the unique priorities and optimization directions of these objective functions. The developed mathematical model defines the decision variables, constraints, and a flexibly adjustable objective function system for this extended MORCMPSP problem.

Key characteristics of the optimization model:

1.1. The introduction of a flexible objective function system allows the consideration of any number of performance indicators with varying dimensions, value ranges, and optimization directions during scheduling.

1.2. In the classical RCPSP model, the resource capacity constraint can only be a time-independent constant value, whereas my RCMOMPSP model ensures the use of arbitrary capacity-time functions.

1.3. Projects can be independent or share common activities. A given activity can belong to multiple projects simultaneously.

1.4. I proved through the solution of well-known scheduling problems that the new model is suitable for handling not only the classical (single-mode RCPSP) project scheduling problems but also the scheduling tasks of discrete manufacturing processes.

The results summarized in Thesis 1 are detailed in Chapter 3 of the dissertation and have been presented in the publication [S19].

Thesis 2: Hybrid scheduling method for solving extended multi-objective, multi-project, resource-constrained scheduling problems.

I developed a combined hybrid solution method for solving extended multi-objective, multi-project, resource-constrained scheduling problems, which divides the problem-solving process into multiple decision phases. In the first level, search algorithms determine the priorities of activities (tasks), and in the second level, constructive algorithms guided by these task priorities generate feasible solutions. The selection among candidate solutions is supported by a model based on multi-objective relative evaluation.

Key characteristics of the method:

2.1. The introduction of a flexible objective function system allows the consideration of any number of performance indicators with varying dimensions, value ranges, and optimization directions during scheduling.

2.2. The method integrally handles constraints related to resources, projects, and activities (tasks).

2.3. The method considers the execution characteristics of projects and tasks defined in the RCMOMPSP model.

2.4. The flexible applicability and efficiency of the hybrid method were validated by the results of test problems solved by the implemented software.

The results summarized in Thesis 2 are detailed in Chapters 4 and 6 of the dissertation and have been presented in the publications [S19, S18, S17, S16, S15].

Thesis 3: Multi-criteria task selection method for constructive scheduling methods.

I developed a multi-criteria task selection method to enhance the flexibility of constructive scheduling methods. Using a relative change-oriented comparison model, this method allows the prioritized simultaneous consideration of many selection criteria, even in cases where the numerical values representing these criteria have mixed signs and support different optimization directions.

Key characteristics of the method:

3.1. One of the most important decision phases of constructive scheduling algorithms is the selection of the next activity (task) to be scheduled from the set of schedulable tasks. The new method allows the simultaneous consideration of more than one selection criterion.

3.2. The selection criteria can include classical task-oriented priority values, dynamically recalculated time-dependent values, and control values representing external management influence.

3.3. The numerical values representing the decision criteria are capable of having both positive and negative signs, and either smaller or larger numerical values can represent the impact of a criterion on the decision.

3.4. Each decision criterion is associated with its own priority value, which regulates the degree of considering the given criterion.

3.5. These extensions together enable the calibration of the selection method by the user.

The results summarized in Thesis 3 are detailed in Subsection 4.2 of the dissertation and have been presented in the publications [S19, S18].

Thesis 4: Combined search metaheuristic for solving permutation sequencing problems.

I developed a metaheuristic search method that simultaneously applies population-based evolutionary and local neighborhood strategies for solving permutation sequencing problems (MOSM). The method is effectively applicable for determining the predictive scheduling and execution order of activities (tasks).

Key characteristics of the method:

4.1. The developed combined search method is advantageously applicable to solve predictive scheduling problems when sufficient time is available to generate and evaluate many solutions.

4.2. The working mode of the solution-modifying operations of the search method is independent of the objective functions.

4.3. The method effectively handles, among others, the problems in which the tasks may have precedence constraints. The built-in normalization algorithm ensures that the sequence complies with all constraints.

4.4. The efficiency of the method was validated by the results obtained on benchmark test problems.

The results summarized in Thesis 4 are detailed in Chapters 4.54.4.1 and 4.5 of the dissertation and have been presented in the publication [S18].

Thesis 5: Integrated recombination and mutation operator to improve the efficiency of the JAYA search metaheuristic.

I developed an operator that simultaneously implements multi-parent recombination and mutation to improve the efficiency of the JAYA search metaheuristic. The new operator (38) is problem-independent, so it is suitable for supporting the solution of any optimization problem. The operator performs more efficiently on the 1||Csum benchmark tasks than the original JAYA operator.

Key characteristics of the method:

5.1. The developed new operator simultaneously implements multi-parent recombination and mutation.

5.2. The working mode of the new operator is independent of objective functions and constraints.

5.3. The efficiency of the new operator was tested on 1||Csum scheduling problems by applying the JAYA search principle to set the priorities of the elementary activities (tasks) to be scheduled. The solution was generated using the serial generation scheme based on the assigned priorities. The results obtained on benchmark test problems demonstrated that the new operator solves problems with fewer search steps than the original JAYA operator.

The results summarized in Thesis 5 are detailed in Sections 4.5.2 and 6.1.2 of the dissertation and have been presented in the publication [S16].

9 Irodalomjegyzék

[1] S. Manning, “Embedding projects in multiple contexts – a structuration perspective,” *International Journal of Project Management,* vol. 26, no. 1, pp. 30-37, 2008.

[2] C. U. Press, “Cambridge Dictionary,” [Online]. Available: https://dictionary.cambridge.org/dictionary/english/project. [Accessed 30 12 2022].

[3] S. H. Huang, J. H. Huang, H. C. Lee and Y. Y. Tong, “A new hybrid algorithm for solving the vehicle routing problem with route balancing,” *International Journal of Industrial Engineering and Management,* vol. 14, no. 1, pp. 51-62, March 2023.

[4] P. Renna, “Workload control order release with controllable processing time policies: an assessment by simulation,” *International Journal of Industrial Engineering and Management,* vol. 13, no. 3, pp. 194-205, Sept. 2022.

[5] M. K. Adeyeri, S. P. Ayodeji, E. O. Olutomilola and O. J. Abayomi, “The Automated Process Control Model for Energy Consumption Optimization within Plantain Flour Processing Facility,” *International Journal of Industrial Engineering and Management,* vol. 13, no. 3, pp. 206-214, Sept. 2022, doi: 10.24867/IJIEM-2022-3-313.

[6] N. Fernandes, M. Thürer, F. Rodrigues, L. Pinto Ferreira, F. J. G. Silva and P. Avila, “Worker Assignment in Dual Resource Constrained Systems Subject to Machine Failures: A Simulation Study,” *International Journal of Industrial Engineering and Management,* vol. 13, no. 2, pp. 110-118, 30 June 2022, doi: 10.24867/IJIEM-2022-2-305.

[7] D. A. Kurniady, Nurochim, A. Komariah, Turwelis, H. T. Hoi and V. H. Ca, “Construction project progress evaluation using a quantitative approach by considering time, cost and quality,” *International Journal of Industrial Engineering and Management,* vol. 13, no. 1, pp. 49-57, March 2022.

[8] S. Gao, J. Daaboul and J. Le Duigou, “Layout and scheduling optimization problem for a reconfigurable manufacturing system,” *International Journal of Industrial Engineering and Management,* vol. 12, no. 3, pp. 174-186, Sept. 2021, doi: 10.24867/IJIEM-2021-3-286.

[9] T. Trisna, M. Marimin, Y. Arkeman and T. C. Sunarti, “Fuzzy multi-objective optimization for wheat flour supply chain considering raw material substitution,” *International Journal of Industrial Engineering and Management,* vol. 11, no. 3, pp. 182-191, Sept. 2020.

[10] S. K. Karimi, S. J. Sadjadi and S. G. J. Naini, “A bi-objective production planning for a flexible supply chain solved using NSGA-II and MOPSO,” *International Journal of Industrial Engineering and Management,* vol. 13, no. 1, pp. 18-37, March 2022, doi: 10.24867/IJIEM-2022-1-298.

[11] N. Tóth and G. Kulcsár, “New models and algorithms to solve integrated problems of production planning and control taking into account worker skills in flexible manufacturing systems,” *International Journal of Industrial Engineering Computations,* vol. 12, no. 4, pp. 381-400, 01 10 2021.

[12] P. Chetthamrongchai, O. Stepanenko, N. Saenko, S. Bakhvalov, G. Aglyamova and A. Iswanto, "A Developed Optimization Model for Mass Production Scheduling Considering the Role of Waste Materials," *International Journal of Industrial Engineering and Management,* vol. 13, no. 2, pp. 135-144, 30 June 2022, doi: 10.24867/IJIEM-2022-2-307.

[13] A. A. B. Pritsker, W. J. Lawrence and P. M. Wolfe, "Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach," *Management Science,* vol. 16, no. 1, pp. 93-108, 1969.

[14] J. Blazewicz, J. K. Lenstra and A. H. Kan, "Scheduling subject to resource constraints: classification and complexity," *Discrete Applied Mathematics,* vol. 5, no. 1, 1983.

[15] R. Kolisch and S. Hartmann, "Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis," *Project Scheduling: Recent Models, Algorithms and Applications,* pp. 147-178, 1999.

[16] S. Hartmann and R. Kolisch, "Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem," *European Journal of Operational Research,* vol. 127, no. 2, pp. 394-407, 2000.

[17] R. Kolisch and S. Hartmann, "PSLIB Single Mode Scheduling Benchmark Dataset," 2006.

[18] R. Pellerin, N. Perrier and F. Berthaut, "A survey of hybrid metaheuristics for the resource-constrained project scheduling problem," *European Journal of Operational Research,* vol. 280, no. 2, pp. 395-416, 2020.

[19] S. Hartmann and D. Briskorn, "An updated survey of variants and extensions of the resource-constrained project scheduling problem," *European Journal of Operational Research,* vol. 297, no. 1, pp. 1-14, 2022.

[20] K. Taha, "Methods That Optimize Multi-Objective Problems: A Survey and Experimental Evaluation," *IEEE Access,* vol. 8, pp. 80855-80878, 2020.

[21] Y.-H. Zhang, Y.-J. Gong, J. Zhang and Y.-b. Ling, "A hybrid evolutionary algorithm with dual populations for many-objective optimization," in *2016 IEEE Congress on Evolutionary Computation (CEC),* 2016.

[22] I. R. Meneghini, M. A. Alves, A. Gaspar-Cunha and F. G. Guimarães, "Scalable and customizable benchmark problems for many-objective optimization," *Applied Soft Computing,* vol. 90, 2020.

[23] H. Dai, W. Cheng and P. Guo, "An Improved Tabu Search for Multi-skill Resource-Constrained Project Scheduling Problems Under Step-Deterioration," *Arabian Journal for Science and Engineering,* vol. 43, no. 6, pp. 3279-3290, 2018.

[24] A. Schnabel, C. Kellenbrink and S. Helber, "Profit-oriented scheduling of resource-constrained projects with flexible capacity constraints," *Business Research,* vol. 11, no. 2, pp. 329-356, 2018.

[25] E. B. Tirkolaee, A. Goli, M. Hematian, A. K. Sangaiah and T. Han, "Multi-objective multi-mode resource constrained project scheduling problem using Pareto-based algorithms," *Computing,* vol. 101, no. 6, pp. 547-570, 2019.

[26] B. H. Tabrizi, "Integrated planning of project scheduling and material procurement considering the environmental impacts," *Computers & Industrial Engineering,* vol. 120, pp. 103-115, 2018.

[27] O. Dridi, S. Krichen and A. Guitouni, "A multiobjective hybrid ant colony optimization approach applied to the assignment and scheduling problem,"

International Transactions in Operational Research, vol. 21, no. 6, pp. 935-953, 2014.

[28] S. Mane and N. M. R. Rao, "Many-objective optimization: Problems and evolutionary algorithms–a short review," *International Journal of Applied Engineering Research,* vol. 12, no. 20, pp. 9774-9793, 2017.

[29] S. U. Manea and M. R. Narsingraoa, "A chaotic-based improved many-objective jaya algorithm for many-objective optimization problems," *International Journal of Industrial Engineering Computations,* vol. 12, no. 1, pp. 49-62, 2021.

[30] G. Kulcsár and F. Erdélyi, "A New Approach to Solve Multi-Objective Scheduling and Rescheduling Tasks," *International Journal of Computational Intelligence Research,* vol. 3, no. 4, pp. 343-351, 01 01 2007.

[31] G. Kulcsár, *Ütemezési modell és heurisztikus módszerek az igény szerinti tömeggyártás finomprogramozásának támogatására, PhD értekezés,* Miskolci Egyetem, 2007.

[32] M. F. Kulcsárné, *Kiterjesztett modellek és módszerek erőforrás-korlátos termelésütemezési feladatok megoldására, PhD értekezés,* Miskolci Egyetem, 2017.

[33] G. Kulcsár and F. Erdélyi, "Modeling and Solving of the Extended Flexible Flow Shop Scheduling Problem," *PRODUCTION SYSTEMS AND INFORMATION ENGINEERING,* vol. 3, pp. 121-139, 2006.

[34] G. Kulcsár, F. Erdélyi and O. Hornyák, "Multi-objective optimization and heuristic approaches for solving scheduling problems.," *MIM'07: Preprints of the IFAC Workshop on Modelling, Management and Control,* pp. 127-132, 14-16 November 2007.

[35] G. Kulcsár and M. F. Kulcsárné, "Detailed production scheduling based on multi-objective search and simulation," *Production Systems and Information Engineering,* vol. 6, pp. 41-56, 2013.

[36] M. F. Kulcsárné and G. Kulcsár, "A new scheduling software for supporting automotive component manufacturing," *LECTURE NOTES IN MECHANICAL ENGINEERING,* pp. 257-274, 2017.

[37] M. F. Kulcsárné and G. Kulcsár, "Modeling and solving an extended parallel resource scheduling problem in the automotive industry," *Acta Polytechnica Hungarica,* vol. 14, no. 4, pp. 27-46, 2017.

[38] G. Kulcsár and M. F. Kulcsárné, "An advanced model for solving industrial scheduling problems," *Production Systems and Information Engineering,* vol. 10, no. 1, pp. 77-89, 2022.

[39] P. Brucker, A. Drexl, R. Möhring, K. Neumann and E. Pesch, "Resource-constrained project scheduling: Notation, classification, models, and methods," *European Journal of Operational Research,* vol. 112, no. 1, pp. 3-41, 1999.

[40] P. Brucker, "Scheduling and constraint propagation," *Discrete Applied Mathematics,* vol. 123, no. 1, pp. 227-256, 2002.

[41] V. L. Tavares, "A review of the contribution of Operational Research to Project Management," *European Journal of Operational Research,* vol. 136, no. 1, pp. 1-18, 2002.

[42] C. Schwindt and J. Zimmermann, Handbook on Project Management and Scheduling Vol.1, 1 ed., Switzerland: Springer International Publishing, 2015, p. 663.

[43] C. Schwindt and J. Zimmermann, Handbook on Project Management and Scheduling Vol. 2, 1 ed., Springer Cham, 2015, p. 740.

[44] S. Hartmann and D. Briskorn, "A survey of variants and extensions of the resource-constrained project scheduling problem," *European Journal of Operational Research,* vol. 207, no. 1, pp. 1-14, 2010.

[45] W. Sun, Y. Pan, X. Lu and Q. Ma, "Research on flexible job-shop scheduling problem based on a modified genetic algorithm," *Journal of Mechanical Science and Technology,* vol. 24, no. 10, pp. 2119-2125, 2010.

[46] K. Ida and K. Oka, "Flexible job-shop scheduling problem by genetic algorithm," *Electrical Engineering in Japan,* vol. 177, no. 3, pp. 28-35, 2011.

[47] M. K. Amjad, S. I. Butt, R. Kousar, R. Ahmad, M. H. Agha, Z. Faping, N. Anjum and U. Asgher, "Recent Research Trends in Genetic Algorithm Based Flexible Job Shop Scheduling Problems," *Mathematical Problems in Engineering,* 2018.

[48] H. C. Fu and P. Liu, "A multi-objective optimization model based on non-dominated sorting genetic algorithm," *International Journal of Simulation Modelling,* vol. 18, no. 3, pp. 510-520, 2019.

[49] Y. Wang, O. Yang and S. N. Wang, "A solution to single-machine inverse job-shop scheduling problem," *International Journal of Simulation Modelling,* vol. 18, no. 1, pp. 335-343, 2019.

[50] M. Yazdani, M. Gholami, M. Zandieh and M. Mousakhani, "A simulated annealing algorithm for flexible job-shop scheduling problem," *Journal of applied sciences,* vol. 9, no. 4, pp. 662-670, 2009.

[51] R. Zhang, "A Simulated Annealing-Based Heuristic Algorithm for Job Shop Scheduling to Minimize Lateness," *International Journal of Advanced Robotic Systems,* vol. 10, no. 4, p. 214, 2013.

[52] W. Bożejko, M. Uchroński and M. Wodecki, "Parallel hybrid metaheuristics for the flexible job shop problem," *Computers & Industrial Engineering,* vol. 59, no. 2, pp. 323-333, 2010.

[53] S. Jia and Z.-H. Hu, "Path-relinking Tabu search for the multi-objective flexible job shop scheduling problem," *Computers & Operations Research,* vol. 47, pp. 11-26, 2014.

[54] M. Ziaee, "Job shop scheduling with makespan objective: A heuristic approach," *International Journal of Industrial Engineering Computations,* vol. 5, no. 2, p. 273–280, 2014.

[55] P. Pongchairerks and V. Kachitvichyanukul, "A particle swarm optimization algorithm on job-shop scheduling problems with multi-purpose machines," *Asia-Pacific Journal of Operational Research,* vol. 26, no. 02, pp. 161-184, 2009.

[56] L. Wang, J. Cai, M. Li and Z. Liu, "Flexible Job Shop Scheduling Problem Using an Improved Ant Colony Optimization," *Scientific Programming,* 2017.

[57] K. Z. Gao, P. N. Suganthan, Q. K. Pan, M. F. Tasgetiren and A. Sadollah, "Artificial bee colony algorithm for scheduling and rescheduling fuzzy flexible job shop problem with new job insertion," *Knowledge-Based Systems,* vol. 109, pp. 1-16, 2016.

[58] G.-C. Luh and S.-W. Lee, "A bacterial evolutionary algorithm for the job shop scheduling problem," *Journal of the Chinese Institute of Industrial Engineers,* vol. 23, no. 3, pp. 185-191, 2006.

[59] H. Zaher, M. El-Sherbieny, N. Ragaa and H. Sayed, “A novel Improved Bat Algorithm for Job Shop,” *International Journal of Computer Applications,* vol. 164, no. 5, pp. 24-30, 2017.

[60] K. C. Udaiyakumar and M. Chandrasekaran, “Application of Firefly Algorithm in Job Shop Scheduling Problem for Minimization of Makespan,” *Procedia Engineering,* vol. 97, pp. 1798-1807, 2014.

[61] K. Jeet, R. Dhir and P. Singh, “Hybrid Black Hole Algorithm for Bi-Criteria Job Scheduling on Parallel Machines,” *International Journal of Intelligent Systems and Applications,* vol. 8, no. 4, pp. 1-17, 2016.

[62] X.-S. Yang, M. Karamanoglu and X. He, “Multi-objective Flower Algorithm for Optimization,” *Procedia Computer Science,* vol. 18, pp. 861-868, 2013.

[63] K. Z. Gao, P. N. Suganthan, Q. K. Pan and M. F. Tasgetiren, “An effective discrete harmony search algorithm for flexible job shop scheduling problem with fuzzy processing time,” *International Journal of Production Research,* vol. 53, no. 19, pp. 5896-5911, 2015.

[64] M. Vanhoucke and J. Coelho, “A matheuristic for the resource-constrained project scheduling problem,” *European Journal of Operational Research,* 2024.

[65] M. Vanhoucke and J. Coelho, “Reducing the feasible solution space of resource-constrained project instances,” *Computers & Operations Research,* vol. 165, 2024.

[66] X. Li, Z. He and N. Wang, “A branch-and-bound algorithm for the proactive resource-constrained project scheduling problem with a robustness maximization objective,” *Computers & Operations Research,* vol. 166, 2024.

[67] A. Knöpfel, B. Gröne and P. Tabeling, Fundamental Modeling Concepts: Effective Communication of IT Systems, Wiley, 2006.

[68] P. Brucker, Scheduling Algorithms, 5 ed., Springer Berlin, Heidelberg, 2007, pp. XII, 371.

[69] S. M. Johnson, “Optimal two- and three-stage production schedules with setup times included,” *Naval Research Logistics Quarterly,* vol. 1, no. 1, pp. 61-68, 1954.

[70] M. L. Pinedo, Planning and Scheduling in Manufacturing and Services, 2 ed., New York: Springer, 2009, pp. XVIII, 536.

[71] K. M. Sallam, R. K. Chakrabortty and M. J. Ryan, “A two-stage multi-operator differential evolution algorithm for solving Resource Constrained Project Scheduling problems,” *Future Generation Computer Systems,* vol. 108, pp. 432-444, July 2020, doi: 10.1016/j.future.2020.02.074.

[72] V. Valls, F. Ballestín and S. Quintanilla, “A hybrid genetic algorithm for the resource-constrained project scheduling problem,” *European Journal of Operational Research,* Vols. 495-508, no. 2, pp. 495-508, 1 March 2008, doi: 10.1016/j.ejor.2006.12.033.

[73] J. C. Rivera, L. F. V. Moreno, F. J. S. Díaz and G. E. Z. Peña, “A HYBRID HEURISTIC ALGORITHM FOR SOLVING THE RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM (RCPSP),” *Revista EIA,* pp. 87-100, Dec. 2013.

[74] F. Berthaut, R. Pellerin, A. Hajji and N. Perrier, “A path relinking-based scatter search for the resource-constrained project scheduling problem,” *International*

Journal of Project Organisation and Management, vol. 10, no. 1, pp. 1-36, 2018, doi: 10.1504/IJPOM.2018.090372.

[75] H. Saad, R. Chakrabortty, S. Elsayed and M. Ryan, "Quantum-Inspired Genetic Algorithm for Resource-Constrained Project-Scheduling," *IEEE Access,* vol. 9, pp. 38488-38502, 26 Febr. 2021, doi: 10.1109/ACCESS.2021.3062790.

[76] H. F. Rahman, R. K. Chakrabortty and M. J. Ryan, "Memetic algorithm for solving resource constrained project scheduling problems," *Automation in Construction,* vol. 111, March 2020, doi: 10.1016/j.autcon.2019.103052.

[77] R.-M. Chen, "Particle swarm optimization with justification and designed mechanisms for resource-constrained project scheduling problem," *Expert Systems with Applications,* vol. 38, no. 6, pp. 7102-7111, June 2011, doi: 10.1016/j.eswa.2010.12.059.

[78] R. Kolisch and S. Hartmann, "Experimental investigation of heuristics for resource-constrained project scheduling: An update," *European Journal of Operational Research,* vol. 174, no. 1, pp. 23-37, 2006.

Saját hivatkozások

[S1] K. Mihály, „Keretrendszer fejlesztése ütemezési feladatok megoldására,” *XV. FMTÜ Nemzetközi Tudományos Konferencia, Műszaki Tudományos Füzetek - FMTÜ 2067-6808 ; volume 15, pp 213-216, 2010*

[S2] K. Mihály and O. Hornyák, „Generic framework for Scheduling problems,” *MicroCAD 2010: XXIV. microCad International Scientific Conference, Section O: Applied information engineering, pp 103-108, 2010*

[S3] K. Mihály, O. Hornyák and Gy. Kulcsár, „Basic concepts of an extended general shop scheduler in manufacturing,” *MicroCAD 2011: XXV. microCad International Scientific Conference, P Section, pp 73-77, 2011*

[S4] K. Mihály and O. Hornyák, „Using graphical processing units in scheduling problems,” *HUNGARIAN JOURNAL OF INDUSTRY AND CHEMISTRY, volume 39, issue 2, pp 215-218, 2011*

[S5] K. Mihály and O. Hornyák, „Grafikus kártya számítási kapacitásának használata determinisztikus ütemezési problémák megoldására,” *MicroCAD 2012: XXVI. microCad International Scientific Conference, Section Applied information engineering, 2012*

[S6] K. Mihály and O. Hornyák, „Grafikus kártya számítási kapacitásának használata determinisztikus, egygépes ütemezési problémák megoldására,” *FMTÜ XVII. Nemzetközi Tudományos Konferencia, Műszaki Tudományos Füzetek - FMTÜ, volume 17, pp 247-250, 2012*

[S7] O. Hornyák and K. Mihály, „Using Graphical Processing Units for Deterministic Single Machine Scheduling Problems,” *PRODUCTION SYSTEMS AND INFORMATION ENGINEERING , volume 6, pp 27-40, 2013*

[S8] K. Mihály, M. Kulcsárné Forrai, and Gy. Kulcsár, „New multi-objective methods to solve resource-constraint multi-project scheduling problems in integrated ERP-APS systems,” *IOP CONFERENCE SERIES: MATERIALS SCIENCE AND ENGINEERING, 448, pp 1-10, Paper 012033, 2018*

[S9] K. Mihály, and Gy. Kulcsár, „Erőforrás-korlátos környezetben párhuzamosan futó projektek többcélú ütemezése,” *MULTIDISZCIPLINÁRIS TUDOMÁNYOK: A MISKOLCI EGYETEM KÖZLEMÉNYE, volume 9, issue 4, pp 301-304, 2019*

[S10] K. Mihály, M. Kulcsárné Forrai, and Gy. Kulcsár, „Új módszerek több projektes, több célfüggvényes, erőforrás-korlátos ütemezési feladatok megoldására integrált vállalatirányítási környezetben,” *Műszaki tudomány az Észak-kelet Magyarországi Régióban 2019 : konferencia előadásai, pp 241-244, 2019*

[S11] Gy. Kulcsár, M. Kulcsárné Forrai, and K. Mihály, „Rugalmas gyártórendszer újraütemezési feladatainak modellezése és megoldása,” *Műszaki tudomány az Észak-kelet Magyarországi Régióban 2019 : konferencia előadásai, pp 189-192, 2019*

[S12] K. Mihály, and Gy. Kulcsár, „Kiterjesztett projektütemezési feladatok megoldása,” *MŰSZAKI TUDOMÁNYOS KÖZLEMÉNYEK, volume 13, pp 137-141, 2020*

[S13] K. Mihály, and Gy. Kulcsár, „Solving Extended Project Scheduling Problems,” *MŰSZAKI TUDOMÁNYOS KÖZLEMÉNYEK, volume 13, issue 1, pp 137-141, 2020*

[S14] K. Mihály, and Gy. Kulcsár, „Párhuzamosan futó projektek többcélú ütemezési modellje és implementálása SAP rendszerben,” *Doktoranduszok Fóruma : Miskolc, 2019. november 21. : Gépészmérnöki és Informatikai Kar Szekciókiadványa, pp 127-13, 2020*

[S15] K. Mihály, M. Kulcsárné Forrai, and Gy. Kulcsár, „Experimental Implementation of a Resource-Constrained Multi-Project Scheduling Problem Solver,” *HUNGARIAN JOURNAL OF INDUSTRY AND CHEMISTRY, volume 49, issue 2, pp 53-58, 2021*

[S16] K. Mihály, M. Kulcsárné Forrai, and Gy. Kulcsár, „Advanced Methods to Solve Multi-project Scheduling Problems Taking into Account Multiple Objective Functions,” *LECTURE NOTES IN MECHANICAL ENGINEERING: Vehicle and Automotive Engineering, volume 4, pp 747-755, 2022, Q4*

[S17] K. Mihály, M. Kulcsárné Forrai, and Gy. Kulcsár, „Multi-objective, multi-project scheduling solver implementation using SAP ABAP language,” *PRODUCTION SYSTEMS AND INFORMATION ENGINEERING, volume 10, issue 1, pp 90-101, 2022*

[S18] K. Mihály, and Gy. Kulcsár, „A New Many-Objective Hybrid Method to Solve Scheduling Problems,” *INTERNATIONAL JOURNAL OF INDUSTRIAL ENGINEERING AND MANAGEMENT, volume 14, issue 4, pp 326-335, 2023, Q2*

[S19] K. Mihály, M. Kulcsárné Forrai, and Gy. Kulcsár, „Solving logistics scheduling problems using an extended many-project optimization model,”, *Advances in Digital Logistics, Logistics and Sustainability, Lecture Notes in Logistics, pp 115 – 144, 2024*

10 Ábra- és táblázatjegyzék

11 Függelék: Többcélú, többprojektes tesztsorozat eredményei

A függelék a 6.3 alfejezetben bemutatott tesztelési folyamat részletes adatait tartalmazza.

11.1 1. tesztcsoport

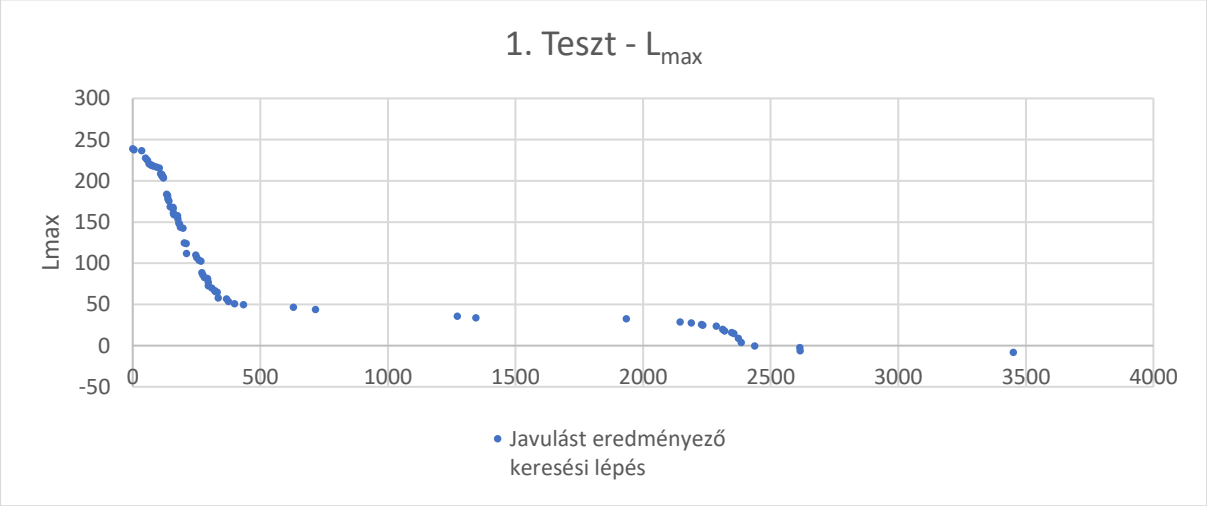
11.1.1 1. teszt

A teszt során az L_{max} célfüggvényt esetén 1 prioritás, minden más célfüggvény esetén 0 prioritás került beállításra. A célfüggvényértékek változását mutatja az alábbi táblázat a keresési lépések előrehaladása során. A táblázatban csak a javulást eredményező lépések szerepelnek.

L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
1	0	0	0	0	0

Javulást eredményező keresési lépés	Projekt portfólió célfüggvény értékei					
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	**239**	239	621	4	286	76193
4	**238**	238	615	4	290	76234
35	**237**	237	622	4	291	76605
50	**228**	228	593	4	294	76982
57	**225**	225	592	4	299	77549
63	**221**	221	608	4	310	77422
69	**220**	220	610	4	305	78262
74	**219**	219	608	4	306	78210
81	**218**	218	601	4	293	77924
92	**217**	217	601	4	292	78016
104	**216**	216	589	4	291	78261
109	**209**	209	563	4	294	79670
113	**208**	208	564	4	298	79375
114	**206**	206	559	4	299	79529
118	**205**	205	543	4	298	80054
120	**204**	204	549	4	298	80003
132	**184**	184	493	4	306	80498
135	**183**	183	486	4	304	80337
136	**182**	182	470	4	306	80489
138	**178**	178	461	4	302	80081
141	**176**	176	473	4	305	80003
146	**169**	169	441	4	310	80339
157	**168**	168	436	4	308	79919
158	**166**	166	413	4	301	79846
159	**161**	161	416	4	302	79528
160	**160**	160	407	4	305	79687
161	**159**	159	418	4	306	80470
175	**158**	158	395	4	305	80192
176	**154**	154	385	4	299	79405
180	**149**	149	366	4	304	79079
182	**148**	148	402	4	300	79204
186	**144**	144	375	4	303	79125
196	**143**	143	359	4	313	80175
202	**125**	125	334	4	314	79751
209	**124**	124	310	4	316	79571
210	**112**	112	260	4	316	80815
246	**110**	110	318	4	307	80127
250	**107**	107	304	4	313	79937
259	**104**	104	309	4	309	79820
267	**103**	103	293	4	306	80347

270	**89**	89	276	4	309	80134
274	**86**	86	274	4	311	80618
281	**83**	83	257	4	312	80737
292	**82**	82	247	4	324	82341
293	**81**	81	238	4	326	82519
295	**77**	77	221	4	321	81671
296	**73**	73	224	4	322	82619
309	**70**	70	209	4	322	81322
321	**67**	67	192	4	322	82249
323	**66**	66	174	4	310	82658
330	**65**	65	188	4	310	82658
334	**58**	58	162	4	308	82587
367	**57**	57	158	4	327	83487
374	**54**	54	163	4	309	83468
398	**51**	51	121	4	328	83647
433	**50**	50	130	4	318	82994
629	**47**	47	138	4	307	82775
716	**44**	44	110	4	306	82876
1271	**36**	36	121	4	326	80370
1344	**34**	34	98	4	308	79709
1934	**33**	33	78	3	309	82080
2144	**29**	29	40	3	313	82628
2189	**28**	28	36	2	314	79521
2229	**26**	26	74	4	322	80667
2234	**25**	25	34	3	324	80595
2286	**24**	24	37	2	310	79453
2311	**20**	20	21	2	306	78949
2319	**18**	18	18	1	310	79438
2346	**16**	16	16	1	304	78712
2355	**15**	15	15	1	312	78487
2373	**9**	9	9	1	306	77543
2384	**4**	4	4	1	304	77949
2437	**0**	0	0	0	302	79557
2614	**-2**	0	0	0	306	80390
2615	**-6**	0	0	0	306	81900
3450	**-8**	0	0	0	321	79707



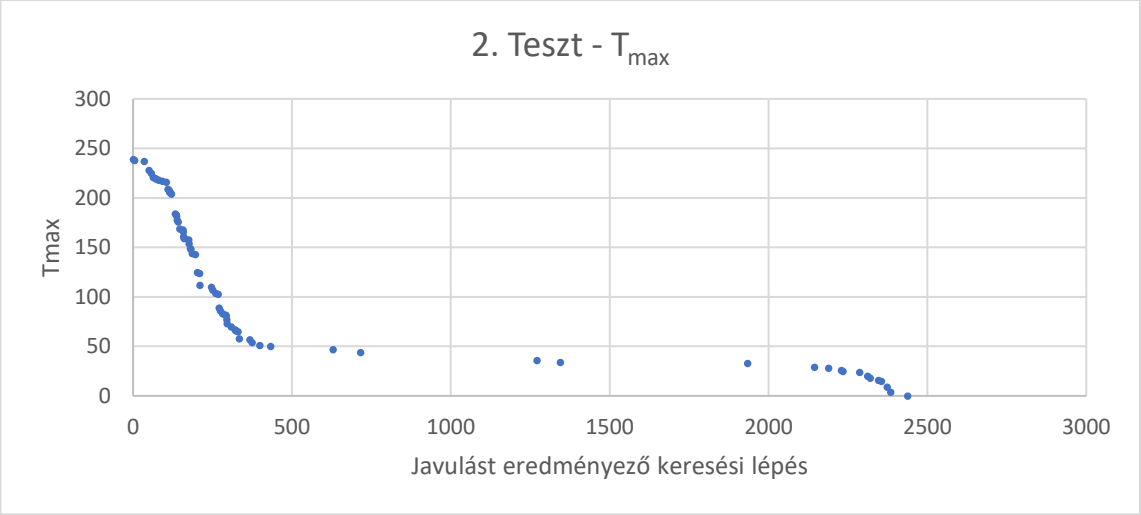
11.1.2 2. Teszt

A teszt során az T_{max} célfüggvényt esetén 1 prioritás, minden más célfüggvény esetén 0 prioritás került beállításra. A célfüggvényértékek változását mutatja az alábbi táblázat a keresési lépések előrehaladása során. A táblázatban csak a javulást eredményező lépések szerepelnek.

L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	1	0	0	0	0

Javulást eredményező keresési lépés	**Projekt portfólió célfüggvény értékek**					
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	239	**239**	621	4	286	76193
4	238	**238**	615	4	290	76234
35	237	**237**	622	4	291	76605
50	228	**228**	593	4	294	76982
57	225	**225**	592	4	299	77549
63	221	**221**	608	4	310	77422
69	220	**220**	610	4	305	78262
74	219	**219**	608	4	306	78210
81	218	**218**	601	4	293	77924
92	217	**217**	601	4	292	78016
104	216	**216**	589	4	291	78261
109	209	**209**	563	4	294	79670
113	208	**208**	564	4	298	79375
114	206	**206**	559	4	299	79529
118	205	**205**	543	4	298	80054
120	204	**204**	549	4	298	80003
132	184	**184**	493	4	306	80498
135	183	**183**	486	4	304	80337
136	182	**182**	470	4	306	80489
138	178	**178**	461	4	302	80081
141	176	**176**	473	4	305	80003
146	169	**169**	441	4	310	80339
157	168	**168**	436	4	308	79919
158	166	**166**	413	4	301	79846
159	161	**161**	416	4	302	79528
160	160	**160**	407	4	305	79687
161	159	**159**	418	4	306	80470
175	158	**158**	395	4	305	80192
176	154	**154**	385	4	299	79405
180	149	**149**	366	4	304	79079
182	148	**148**	402	4	300	79204
186	144	**144**	375	4	303	79125
196	143	**143**	359	4	313	80175
202	125	**125**	334	4	314	79751
209	124	**124**	310	4	316	79571
210	112	**112**	260	4	316	80815
246	110	**110**	318	4	307	80127
250	107	**107**	304	4	313	79937
259	104	**104**	309	4	309	79820
267	103	**103**	293	4	306	80347
270	89	**89**	276	4	309	80134
274	86	**86**	274	4	311	80618
281	83	**83**	257	4	312	80737
292	82	**82**	247	4	324	82341
293	81	**81**	238	4	326	82519
295	77	**77**	221	4	321	81671

296	73	**73**	224	4	322	82619
309	70	**70**	209	4	322	81322
321	67	**67**	192	4	322	82249
323	66	**66**	174	4	310	82658
330	65	**65**	188	4	310	82658
334	58	**58**	162	4	308	82587
367	57	**57**	158	4	327	83487
374	54	**54**	163	4	309	83468
398	51	**51**	121	4	328	83647
433	50	**50**	130	4	318	82994
629	47	**47**	138	4	307	82775
716	44	**44**	110	4	306	82876
1271	36	**36**	121	4	326	80370
1344	34	**34**	98	4	308	79709
1934	33	**33**	78	3	309	82080
2144	29	**29**	40	3	313	82628
2189	28	**28**	36	2	314	79521
2229	26	**26**	74	4	322	80667
2234	25	**25**	34	3	324	80595
2286	24	**24**	37	2	310	79453
2311	20	**20**	21	2	306	78949
2319	18	**18**	18	1	310	79438
2346	16	**16**	16	1	304	78712
2355	15	**15**	15	1	312	78487
2373	9	**9**	9	1	306	77543
2384	4	**4**	4	1	304	77949
2437	0	**0**	0	0	302	79557



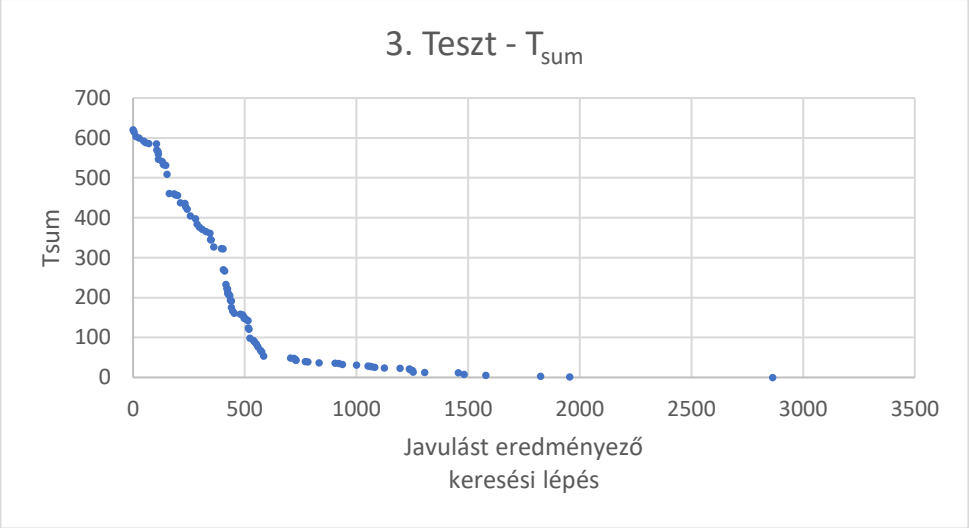
11.1.3 3. Teszt

A teszt során az T_{sum} célfüggvényt esetén 1 prioritás, minden más célfüggvény esetén 0 prioritás került beállításra. A célfüggvényértékek változását mutatja az alábbi táblázat a keresési lépések előrehaladása során. A táblázatban csak a javulást eredményező lépések szerepelnek.

L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	0	1	0	0	0

Javulást eredményező keresési lépés	**Projekt portfólió célfüggvény értékek**					
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	239	239	**621**	4	286	76193
4	238	238	**615**	4	290	76234
13	226	226	**604**	4	298	77681
26	224	224	**600**	4	296	77453
47	228	228	**593**	4	299	78264
55	227	227	**589**	4	298	78127
69	226	226	**586**	4	297	78089
104	239	239	**585**	4	313	80302
106	238	238	**570**	4	312	80786
111	237	237	**566**	4	311	79791
112	239	239	**560**	4	312	80067
113	237	237	**547**	4	311	80129
130	235	235	**541**	4	309	80408
135	231	231	**533**	4	306	80437
145	232	232	**532**	4	319	80726
151	207	207	**509**	4	307	80438
161	165	165	**461**	4	313	80285
183	165	165	**460**	4	313	80111
188	165	165	**458**	4	313	80131
199	163	163	**456**	4	315	80392
211	163	163	**438**	4	307	80402
231	163	163	**436**	4	304	80378
235	166	166	**428**	4	310	80597
242	172	172	**422**	4	313	80743
256	156	156	**405**	4	313	80326
279	152	152	**398**	4	316	79848
286	156	156	**384**	4	320	81835
296	155	155	**377**	4	303	81541
309	148	148	**371**	4	313	81577
326	148	148	**366**	4	312	81546
344	167	167	**362**	4	325	80928
346	147	147	**346**	4	310	81400
349	142	142	**345**	4	330	81656
361	141	141	**327**	4	319	81842
395	138	138	**323**	4	311	81933
402	137	137	**322**	4	312	82002
404	147	147	**270**	4	309	80969
410	135	135	**267**	4	314	81159
415	134	134	**233**	4	308	80046
419	142	142	**223**	3	311	80108
422	142	142	**222**	3	308	80396
423	141	141	**213**	3	304	79597
424	140	140	**210**	3	304	80220
429	140	140	**207**	3	307	80921
433	139	139	**205**	3	308	80661
435	134	134	**194**	3	311	79871

437	112	112	**193**	3	313	81167
439	129	129	**192**	3	307	80006
440	110	110	**176**	3	309	80161
445	107	107	**167**	3	307	79901
452	108	108	**161**	3	308	80343
479	106	106	**159**	3	314	80211
489	104	104	**157**	3	316	80278
496	102	102	**152**	3	324	80985
497	103	103	**148**	3	322	79956
505	100	100	**146**	3	325	80494
514	90	90	**143**	3	317	79275
516	91	91	**124**	3	326	80686
518	90	90	**121**	3	325	80482
522	83	83	**99**	3	324	80162
539	81	81	**93**	3	326	81275
543	86	86	**90**	3	324	79565
551	84	84	**84**	1	324	80945
559	64	64	**77**	2	328	81158
570	64	64	**68**	2	323	80990
576	64	64	**64**	1	321	81599
584	54	54	**54**	1	325	82412
705	49	49	**49**	1	346	82859
721	48	48	**48**	1	343	82735
724	46	46	**46**	1	323	81774
728	44	44	**44**	1	324	81057
729	43	43	**43**	1	334	82187
770	40	40	**40**	1	330	82389
782	39	39	**39**	1	320	81441
832	37	37	**37**	1	334	80757
904	36	36	**36**	1	326	81317
921	35	35	**35**	1	336	80670
937	33	33	**33**	1	342	79798
1000	31	31	**31**	1	331	81300
1052	29	29	**29**	1	326	80180
1062	28	28	**28**	1	309	81331
1067	27	27	**27**	1	321	80138
1082	26	26	**26**	1	317	81964
1125	24	24	**24**	1	310	80842
1195	23	23	**23**	1	311	81356
1236	22	22	**22**	1	312	81797
1239	20	20	**20**	1	314	81421
1251	18	18	**18**	1	301	80938
1254	14	14	**14**	1	318	81509
1305	13	13	**13**	1	311	82019
1455	12	12	**12**	1	328	82852
1481	8	8	**8**	1	321	81832
1579	6	6	**6**	1	314	81979
1824	3	3	**3**	1	333	82306
1955	2	2	**2**	1	318	81355
2863	0	0	**0**	0	311	80470

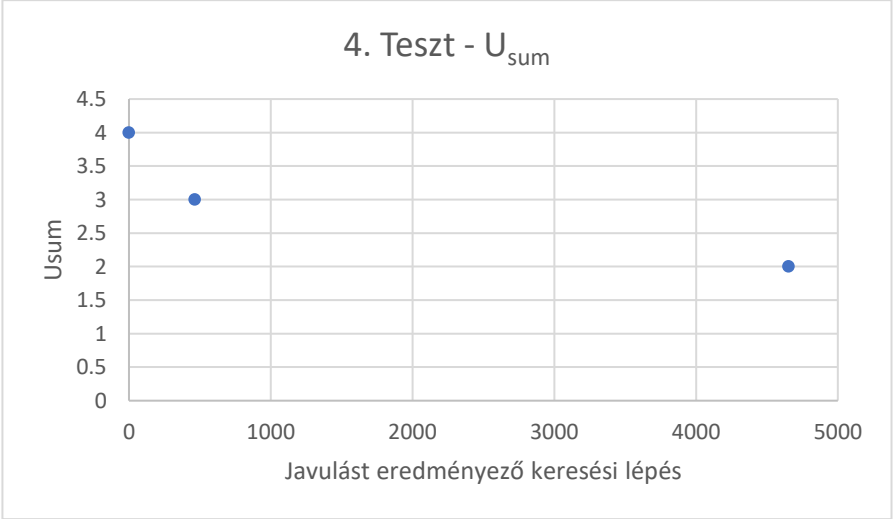


11.1.4 4. Teszt

A teszt során az U_{sum} célfüggvényt esetén 1 prioritás, minden más célfüggvény esetén 0 prioritás került beállításra. A célfüggvényértékek változását mutatja az alábbi táblázat a keresési lépések előrehaladása során. A táblázatban csak a javulást eredményező lépések szerepelnek.

L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	0	0	1	0	0

Javulást eredményező keresési lépés	**Projekt portfólió célfüggvény értékek**					
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	239	239	621	**4**	286	76193
465	227	227	437	**3**	315	84239
4654	256	256	436	**2**	323	84605

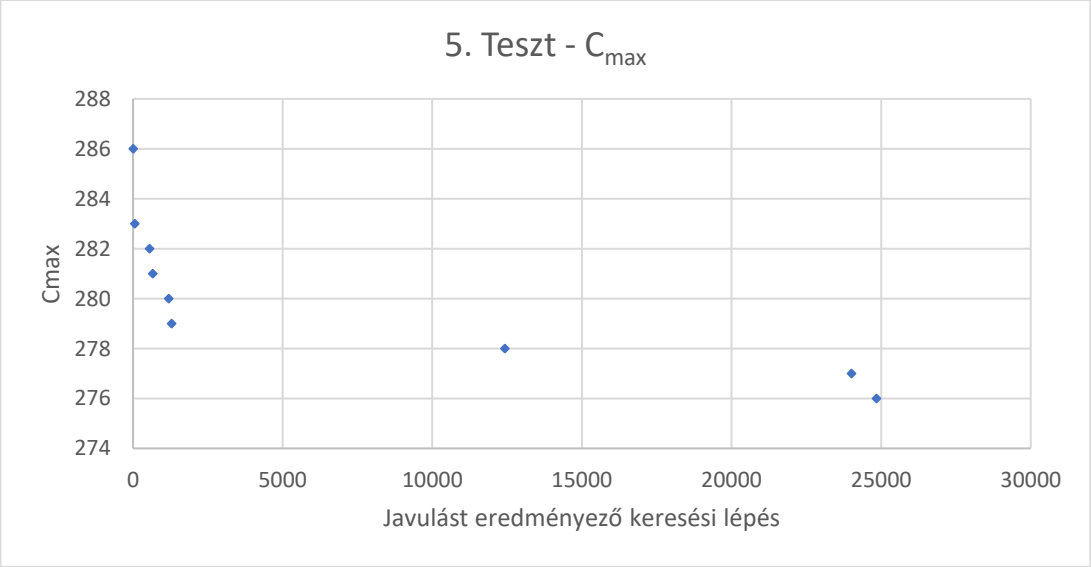


11.1.5 5. Teszt

A teszt során az C_{max} célfüggvényt esetén 1 prioritás, minden más célfüggvény esetén 0 prioritás került beállításra. A célfüggvényértékek változását mutatja az alábbi táblázat a keresési lépések előrehaladása során. A táblázatban csak a javulást eredményező lépések szerepelnek.

L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	0	0	0	1	0

Javulást eredményező keresési lépés	**Projekt portfólió célfüggvény értékek**					
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	239	239	621	4	**286**	76193
58	244	244	617	4	**283**	77027
550	242	242	625	4	**282**	78570
652	238	238	644	4	**281**	79219
1194	203	203	631	4	**280**	79805
1287	204	204	629	4	**279**	79873
12425	207	207	623	4	**278**	80081
24008	236	236	666	4	**277**	80602
24845	237	237	637	4	**276**	79837



11.1.6 6. Teszt

A teszt során az C_{sum} célfüggvényt esetén 1 prioritás, minden más célfüggvény esetén 0 prioritás került beállításra. A célfüggvényértékek változását mutatja az alábbi táblázat a keresési lépések előrehaladása során. A táblázatban csak a javulást eredményező lépések szerepelnek.

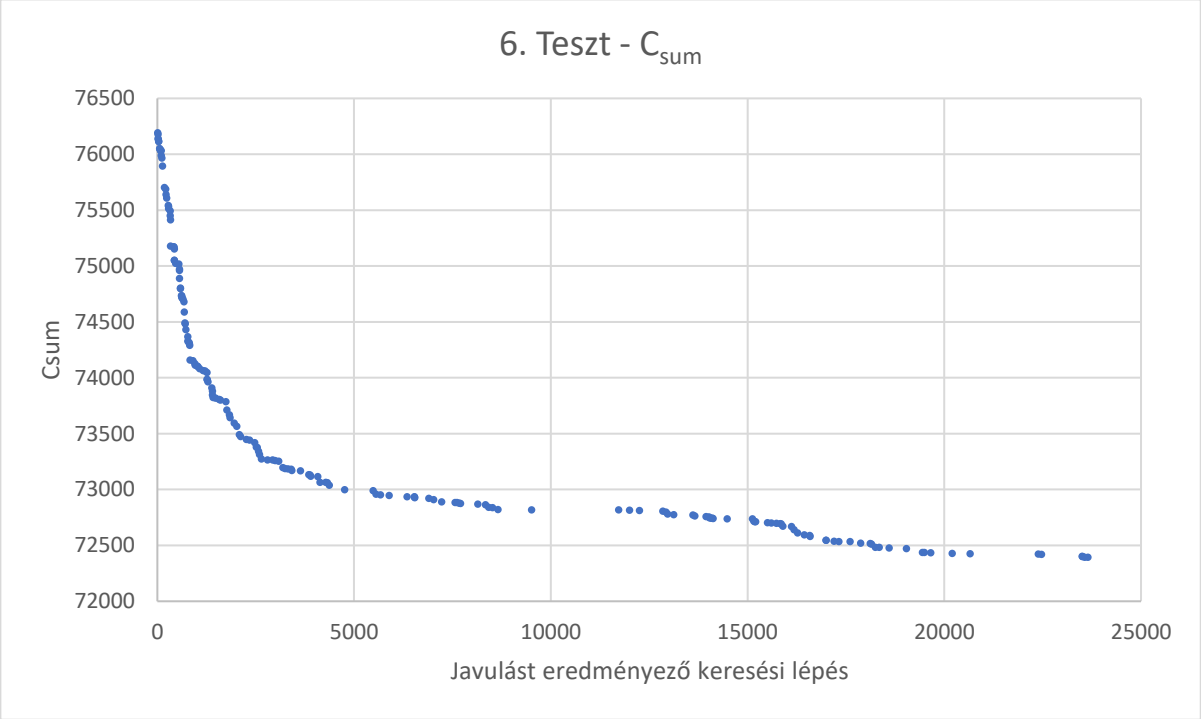
L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	0	0	0	0	1

Javulást eredményező keresési lépés	Projekt portfólió célfüggvény értékek					
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	239	239	621	4	286	**76193**
7	239	239	617	4	286	**76178**
9	239	239	622	4	286	**76141**
17	239	239	622	4	286	**76131**
24	240	240	625	4	287	**76116**
29	241	241	626	4	288	**76115**
51	245	245	637	4	292	**76052**
69	244	244	634	4	291	**76041**
70	244	244	634	4	291	**76037**
72	243	243	598	4	290	**76036**
88	243	243	598	4	290	**76032**
90	242	242	598	4	306	**75989**
104	241	241	596	4	305	**75965**
124	241	241	586	4	305	**75895**
172	251	251	642	4	298	**75703**
203	251	251	642	4	298	**75687**
215	251	251	640	4	298	**75640**
226	251	251	628	4	298	**75607**
269	250	250	629	4	298	**75542**
275	248	248	623	4	296	**75534**
277	249	249	630	4	297	**75513**
315	253	253	621	4	300	**75493**
322	248	248	608	4	296	**75451**
323	252	252	612	4	303	**75414**
325	251	251	603	4	302	**75179**
408	251	251	605	4	302	**75173**
419	251	251	605	4	302	**75171**
421	252	252	608	4	303	**75153**
422	251	251	600	4	290	**75051**
424	251	251	600	4	290	**75049**
455	251	251	600	4	290	**75024**
538	257	257	591	4	296	**75018**
552	251	251	600	4	290	**74972**
555	251	251	619	4	290	**74962**
557	257	257	591	4	296	**74890**
575	250	250	615	4	289	**74804**
582	250	250	616	4	289	**74796**
606	250	250	616	4	289	**74735**
612	255	255	594	4	302	**74734**
615	255	255	594	4	302	**74722**
620	255	255	590	4	302	**74713**
643	255	255	590	4	302	**74704**

672	254	254	582	4	301	**74680**
675	253	253	579	4	300	**74587**
690	253	253	607	4	300	**74492**
701	249	249	599	4	300	**74481**
721	234	234	588	4	309	**74430**
763	247	247	594	4	298	**74367**
768	247	247	594	4	298	**74327**
799	249	249	599	4	297	**74317**
808	248	248	603	4	297	**74306**
817	250	250	621	4	299	**74289**
825	249	249	608	4	298	**74159**
892	249	249	620	4	298	**74152**
945	243	243	612	4	295	**74128**
951	243	243	612	4	295	**74114**
1006	243	243	613	4	295	**74104**
1025	243	243	613	4	295	**74099**
1064	235	235	621	4	306	**74080**
1157	235	235	622	4	306	**74065**
1202	235	235	622	4	306	**74060**
1251	235	235	622	4	306	**74047**
1255	200	200	584	4	312	**73987**
1261	200	200	584	4	312	**73981**
1280	200	200	578	4	306	**73965**
1378	240	240	620	4	303	**73908**
1389	237	237	612	4	300	**73881**
1395	235	235	608	4	299	**73846**
1405	235	235	608	4	299	**73823**
1470	235	235	605	4	298	**73817**
1560	233	233	591	4	300	**73805**
1595	233	233	591	4	300	**73800**
1735	233	233	591	4	300	**73785**
1756	227	227	587	4	296	**73711**
1823	227	227	587	4	296	**73668**
1838	227	227	587	4	296	**73644**
1942	226	226	581	4	292	**73594**
2007	226	226	581	4	292	**73566**
2079	226	226	581	4	292	**73492**
2108	203	203	563	4	297	**73475**
2255	203	203	547	4	297	**73449**
2333	203	203	547	4	297	**73441**
2469	203	203	547	4	297	**73420**
2510	203	203	547	4	297	**73379**
2534	203	203	547	4	297	**73377**
2563	203	203	547	4	297	**73343**
2589	203	203	545	4	294	**73313**
2641	203	203	545	4	294	**73274**
2793	203	203	545	4	294	**73265**
2925	203	203	545	4	294	**73264**
2986	203	203	549	4	294	**73259**
3074	203	203	549	4	294	**73253**
3181	203	203	549	4	294	**73197**
3241	203	203	549	4	294	**73187**
3311	203	203	549	4	294	**73185**
3396	203	203	549	4	294	**73183**
3408	203	203	549	4	294	**73170**
3632	203	203	549	4	294	**73166**
3839	203	203	549	4	294	**73132**
3875	203	203	549	4	294	**73130**

3888	203	203	549	4	294	**73118**
4067	203	203	549	4	294	**73117**
4124	203	203	549	4	294	**73064**
4270	203	203	549	4	294	**73063**
4310	203	203	549	4	294	**73062**
4365	203	203	541	4	294	**73039**
4750	205	205	544	4	294	**72997**
5480	205	205	544	4	294	**72991**
5546	205	205	544	4	294	**72959**
5661	205	205	544	4	294	**72952**
5887	205	205	546	4	294	**72947**
6340	205	205	546	4	294	**72936**
6526	205	205	546	4	294	**72935**
6528	205	205	546	4	294	**72932**
6535	205	205	546	4	294	**72927**
6893	205	205	546	4	294	**72920**
7012	205	205	546	4	294	**72909**
7217	205	205	546	4	294	**72890**
7558	205	205	546	4	294	**72885**
7605	205	205	546	4	294	**72884**
7661	205	205	546	4	294	**72879**
7699	205	205	546	4	294	**72875**
8138	205	205	546	4	294	**72868**
8330	205	205	554	4	294	**72863**
8417	205	205	546	4	294	**72840**
8513	205	205	546	4	294	**72839**
8650	205	205	540	4	294	**72822**
9500	205	205	540	4	294	**72819**
11714	205	205	540	4	294	**72817**
11990	205	205	540	4	294	**72816**
12243	205	205	540	4	294	**72813**
12838	244	244	572	4	298	**72807**
12921	244	244	573	4	298	**72797**
12965	230	230	583	4	303	**72781**
13117	242	242	574	4	296	**72774**
13600	242	242	574	4	296	**72771**
13653	242	242	574	4	296	**72763**
13935	242	242	574	4	296	**72758**
14008	242	242	574	4	296	**72755**
14028	242	242	574	4	296	**72745**
14069	242	242	574	4	296	**72743**
14099	242	242	574	4	296	**72742**
14114	234	234	566	4	296	**72740**
14471	234	234	566	4	296	**72739**
15119	234	234	566	4	296	**72738**
15149	234	234	566	4	296	**72719**
15177	234	234	566	4	296	**72713**
15198	234	234	566	4	296	**72711**
15497	234	234	566	4	296	**72704**
15600	224	224	572	4	296	**72701**
15731	224	224	572	4	296	**72697**
15822	224	224	564	4	297	**72696**
15849	224	224	564	4	297	**72692**
15892	224	224	564	4	297	**72671**
16111	224	224	564	4	297	**72668**
16178	219	219	552	4	299	**72639**
16266	219	219	549	4	293	**72611**
16435	219	219	549	4	292	**72593**

16573	219	219	549	4	292	**72589**
16585	219	219	549	4	292	**72579**
16986	219	219	553	4	290	**72547**
16988	219	219	553	4	290	**72545**
17188	218	218	547	4	289	**72537**
17314	218	218	547	4	289	**72535**
17599	218	218	548	4	289	**72533**
17870	218	218	548	4	291	**72519**
18114	228	228	548	4	290	**72516**
18144	228	228	548	4	290	**72510**
18245	228	228	548	4	290	**72484**
18342	228	228	548	4	290	**72482**
18593	228	228	548	4	290	**72476**
19032	228	228	548	4	290	**72471**
19439	228	228	548	4	290	**72438**
19487	228	228	548	4	290	**72437**
19646	227	227	548	4	289	**72435**
20197	227	227	548	4	289	**72427**
20652	227	227	548	4	289	**72424**
22386	227	227	548	4	289	**72422**
22461	227	227	548	4	289	**72420**
23493	227	227	548	4	289	**72402**
23507	227	227	548	4	289	**72400**
23565	227	227	548	4	289	**72395**
23647	227	227	548	4	289	**72393**



11.2 2. tesztcsoport

11.2.1 7. Teszt

A teszt során minden célfüggvényt esetén 1 prioritás került beállításra. A célfüggvényértékek változását mutatja az alábbi táblázat a keresési lépések előrehaladása során. A táblázatban csak a javulást eredményező lépések szerepelnek.

L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
1	1	1	1	1	1

Javulást eredményező keresési lépés	Projekt portfólió célfüggvény értékek					
	L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	239	239	621	4	286	76193
4	238	238	615	4	290	76234
10	238	238	615	4	290	76192
15	236	236	610	4	289	76163
20	217	217	597	4	288	76214
22	217	217	597	4	288	76208
24	216	216	597	4	287	76281
31	216	216	597	4	287	76261
44	216	216	590	4	290	76161
50	216	216	590	4	290	76130
56	215	215	587	4	290	76078
60	215	215	584	4	290	76054
64	215	215	583	4	290	76090
65	215	215	581	4	290	76042
70	215	215	583	4	284	76455
80	215	215	583	4	284	76453
93	211	211	581	4	290	76336
95	211	211	577	4	291	76481
96	211	211	577	4	291	76448
103	211	211	574	4	291	76399
104	202	202	573	4	289	76713
105	201	201	567	4	294	76766
107	201	201	568	4	294	76589
111	201	201	566	4	294	76644
114	201	201	573	4	289	76968
115	201	201	573	4	289	76938
120	201	201	573	4	289	76936
124	201	201	569	4	288	76731
128	201	201	569	4	287	76729
141	197	197	565	4	295	77202
142	197	197	565	4	295	77187
143	197	197	564	4	295	76826
146	195	195	563	4	296	77104
147	195	195	566	4	294	77114
153	195	195	566	4	294	76974
156	195	195	559	4	288	76921
158	195	195	558	4	288	76862
160	195	195	552	4	288	77164
162	188	188	537	4	295	77697
164	188	188	537	4	295	77512
165	184	184	535	4	297	78508

166	184	184	536	4	297	78313
167	184	184	534	4	297	78113
169	169	169	535	4	297	78153
171	168	168	523	4	299	78159
175	164	164	486	4	294	77756
178	164	164	482	4	294	77980
179	158	158	497	4	299	78996
182	151	151	464	4	308	78565
184	149	149	462	4	303	78573
190	147	147	427	4	303	78309
193	142	142	430	4	305	78256
201	133	133	437	4	313	78414
202	132	132	402	4	309	77934
203	132	132	402	4	309	77717
213	132	132	398	4	309	77996
220	132	132	398	4	309	77976
222	132	132	398	4	309	77921
226	131	131	398	4	311	77779
227	131	131	387	4	311	78602
229	127	127	383	4	309	79126
231	125	125	394	4	305	78853
232	124	124	369	4	307	78871
235	124	124	369	4	307	78827
237	122	122	351	4	304	78152
260	122	122	352	4	299	78157
279	119	119	341	4	308	78520
282	115	115	341	4	309	78802
283	114	114	331	4	313	78980
284	114	114	329	4	312	79359
286	114	114	317	4	319	78652
287	111	111	309	4	317	78553
288	107	107	319	4	310	77673
290	105	105	293	4	317	77822
300	105	105	290	4	313	78151
304	105	105	278	4	312	78103
318	102	102	286	4	311	78749
319	99	99	274	4	316	79111
323	99	99	263	4	310	78595
326	99	99	250	4	309	78587
327	99	99	250	4	309	78541
344	99	99	250	4	309	78476
348	97	97	243	4	327	79063
355	97	97	240	4	323	79280
357	97	97	240	4	323	79233
361	99	99	228	4	326	79215
368	100	100	206	4	332	79551
369	100	100	197	4	324	79418
375	100	100	197	4	324	79408
377	98	98	195	4	324	79312
378	93	93	195	4	325	79459
379	94	94	189	4	313	80146
380	90	90	174	4	327	81353
381	84	84	143	3	313	79834
390	83	83	141	3	312	80188
404	78	78	140	3	324	79411
405	78	78	130	3	310	78891
410	73	73	105	3	309	78030
419	63	63	98	3	328	80935

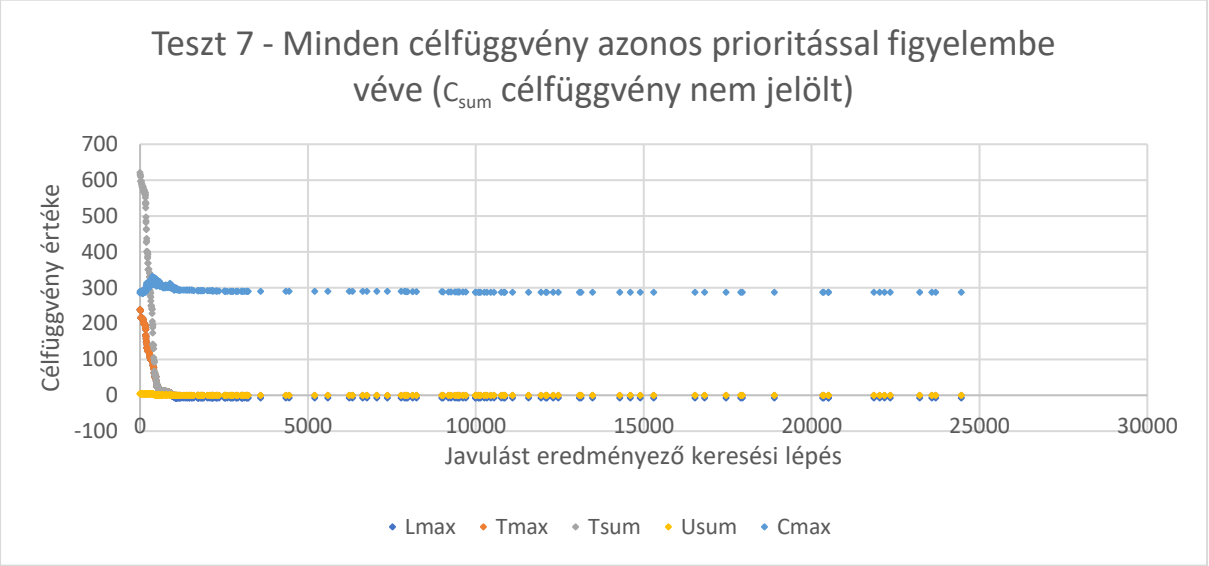
426	63	63	95	3	321	80176
427	63	63	92	3	316	79810
428	61	61	68	2	315	80108
433	51	51	60	3	316	80799
440	60	60	63	2	315	79870
441	51	51	58	2	317	80588
447	54	54	54	1	327	80875
452	54	54	54	1	327	80856
455	53	53	53	1	319	81040
456	53	53	53	1	319	80918
464	53	53	53	1	319	80867
465	53	53	53	1	320	80361
466	53	53	53	1	320	80353
470	53	53	53	1	316	80858
472	52	52	52	1	318	80341
474	44	44	44	1	315	79462
479	39	39	39	1	315	79845
491	35	35	35	1	304	78752
492	32	32	32	1	315	79347
494	30	30	30	1	315	79611
497	30	30	30	1	314	79504
501	30	30	30	1	310	78609
503	27	27	27	1	313	79547
504	27	27	27	1	310	79334
505	24	24	24	1	321	80033
507	24	24	24	1	321	79901
508	24	24	24	1	313	78493
512	24	24	24	1	313	78437
515	24	24	24	1	312	78359
530	24	24	24	1	309	78165
535	23	23	23	1	312	79517
537	23	23	23	1	312	79494
538	23	23	23	1	309	78652
545	23	23	23	1	308	78516
546	20	20	20	1	317	79794
547	17	17	17	1	315	79309
549	15	15	15	1	313	79776
551	15	15	15	1	306	78853
578	14	14	14	1	318	79591
580	14	14	14	1	315	79262
582	14	14	14	1	312	79461
584	14	14	14	1	309	79572
585	14	14	14	1	309	79498
588	14	14	14	1	309	79366
590	14	14	14	1	305	78686
591	14	14	14	1	304	77729
601	14	14	14	1	304	77715
634	13	13	13	1	311	79185
639	13	13	13	1	308	79027
643	13	13	13	1	309	78485
651	13	13	13	1	303	78838
664	13	13	13	1	303	78456
666	13	13	13	1	303	78449
670	13	13	13	1	303	78302
672	13	13	13	1	303	78288
673	13	13	13	1	303	78262
674	13	13	13	1	303	78256
682	13	13	13	1	303	78182

683	13	13	13	1	300	78121
703	13	13	13	1	300	77550
750	13	13	13	1	300	77545
759	11	11	11	1	307	79871
763	11	11	11	1	307	79748
766	11	11	11	1	306	79605
769	11	11	11	1	306	79574
771	11	11	11	1	306	79570
774	11	11	11	1	306	79556
776	11	11	11	1	307	78929
777	11	11	11	1	307	78655
794	11	11	11	1	304	78521
800	11	11	11	1	302	78675
804	10	10	10	1	301	78239
845	10	10	10	1	301	78193
863	10	10	10	1	301	77811
888	6	6	6	1	312	78748
892	6	6	6	1	307	78634
895	6	6	6	1	307	78346
896	6	6	6	1	301	77963
897	6	6	6	1	301	77922
899	6	6	6	1	301	77921
924	6	6	6	1	300	77933
927	6	6	6	1	299	78002
935	6	6	6	1	299	77991
946	3	3	3	1	305	78531
949	3	3	3	1	305	78378
950	3	3	3	1	302	78781
951	-2	0	0	0	307	78890
952	-2	0	0	0	302	78607
960	-2	0	0	0	302	78523
965	-2	0	0	0	302	78372
966	-2	0	0	0	301	78352
970	-2	0	0	0	300	78486
977	-2	0	0	0	300	78376
979	-2	0	0	0	299	78350
983	-2	0	0	0	299	78260
984	-2	0	0	0	300	77628
988	-2	0	0	0	300	77626
991	-2	0	0	0	300	77491
998	-2	0	0	0	299	77606
1013	-2	0	0	0	299	77510
1017	-2	0	0	0	295	77271
1019	-2	0	0	0	294	77076
1020	-2	0	0	0	294	76934
1027	-6	0	0	0	296	77750
1039	-6	0	0	0	297	77391
1040	-6	0	0	0	297	77278
1041	-6	0	0	0	297	77118
1054	-6	0	0	0	297	77058
1056	-7	0	0	0	300	77900
1057	-7	0	0	0	300	77874
1062	-7	0	0	0	295	77110
1066	-7	0	0	0	295	77101
1071	-7	0	0	0	298	76284
1085	-7	0	0	0	298	76225
1089	-7	0	0	0	298	76219
1090	-7	0	0	0	298	76030

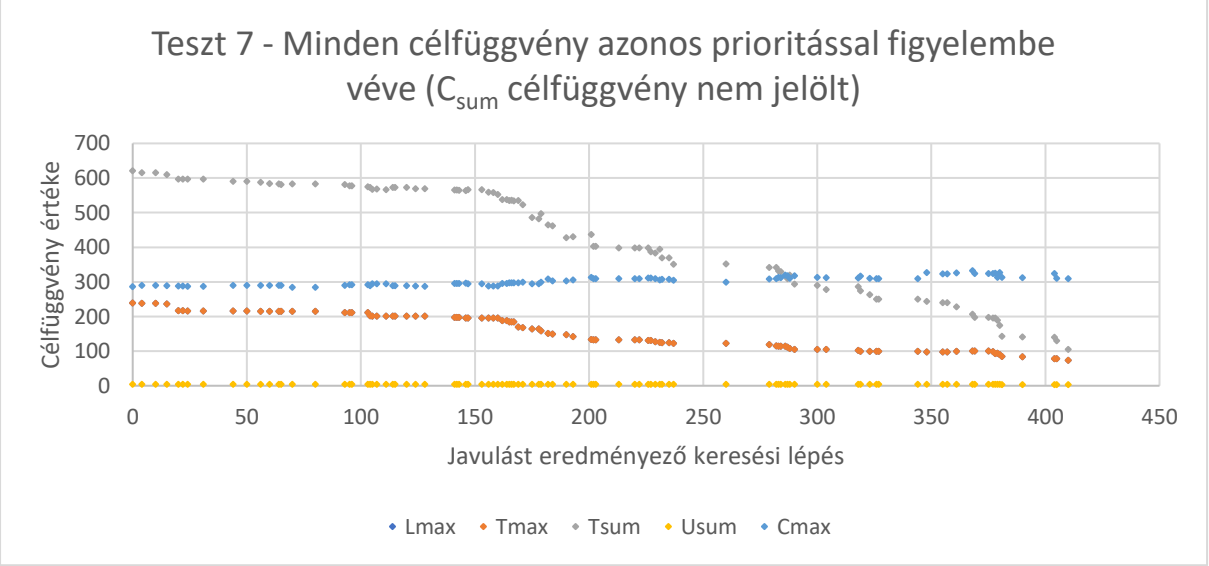
1091	-7	0	0	0	296	76108
1094	-7	0	0	0	296	76082
1101	-7	0	0	0	296	75756
1111	-7	0	0	0	294	75850
1155	-7	0	0	0	294	75701
1160	-7	0	0	0	294	75686
1183	-7	0	0	0	294	75650
1185	-7	0	0	0	294	75621
1219	-7	0	0	0	294	75588
1282	-7	0	0	0	294	75519
1345	-7	0	0	0	294	75509
1388	-7	0	0	0	294	75477
1413	-7	0	0	0	294	75422
1468	-7	0	0	0	294	75411
1473	-7	0	0	0	294	75390
1541	-7	0	0	0	294	75377
1543	-7	0	0	0	294	75325
1600	-7	0	0	0	292	75333
1722	-7	0	0	0	292	75330
1723	-7	0	0	0	292	75326
1729	-7	0	0	0	292	75320
1735	-7	0	0	0	292	75307
1748	-7	0	0	0	292	75156
1796	-7	0	0	0	292	75055
1820	-7	0	0	0	292	75050
1843	-7	0	0	0	292	75042
1861	-7	0	0	0	292	75025
1892	-7	0	0	0	292	75003
2010	-7	0	0	0	292	75001
2068	-7	0	0	0	292	74937
2124	-7	0	0	0	292	74850
2148	-7	0	0	0	292	74781
2150	-7	0	0	0	292	74765
2164	-7	0	0	0	291	74777
2193	-7	0	0	0	291	74731
2269	-7	0	0	0	291	74728
2273	-7	0	0	0	291	74725
2297	-7	0	0	0	291	74722
2347	-7	0	0	0	291	74701
2512	-7	0	0	0	291	74694
2533	-7	0	0	0	290	74894
2539	-7	0	0	0	290	74893
2595	-7	0	0	0	290	74847
2698	-7	0	0	0	290	74843
2704	-7	0	0	0	290	74768
2746	-7	0	0	0	290	74724
2772	-7	0	0	0	290	74714
2868	-7	0	0	0	290	74684
2869	-7	0	0	0	290	74668
2917	-7	0	0	0	290	74662
3023	-7	0	0	0	290	74520
3034	-7	0	0	0	290	74435
3042	-7	0	0	0	290	74423
3049	-7	0	0	0	290	74413
3100	-7	0	0	0	290	74409
3173	-7	0	0	0	290	74405
3182	-7	0	0	0	290	74366
3209	-7	0	0	0	290	74362

3222	-7	0	0	0	290	74361
3594	-7	0	0	0	290	74360
4347	-7	0	0	0	290	74357
4436	-7	0	0	0	290	74352
5205	-7	0	0	0	290	74345
5592	-7	0	0	0	290	74308
6235	-7	0	0	0	290	74304
6330	-7	0	0	0	290	74303
6627	-7	0	0	0	290	74271
6756	-7	0	0	0	290	74268
7051	-7	0	0	0	290	74234
7361	-7	0	0	0	290	74210
7771	-7	0	0	0	290	74208
7877	-7	0	0	0	290	74207
7915	-7	0	0	0	289	74291
7964	-7	0	0	0	289	74287
8117	-7	0	0	0	289	74283
8236	-7	0	0	0	289	74282
8982	-7	0	0	0	289	74281
9022	-7	0	0	0	288	74467
9185	-7	0	0	0	288	74459
9260	-7	0	0	0	288	74450
9374	-7	0	0	0	288	74445
9442	-7	0	0	0	288	74431
9483	-7	0	0	0	288	74396
9513	-7	0	0	0	288	74385
9617	-7	0	0	0	288	74381
9699	-7	0	0	0	288	74366
9989	-7	0	0	0	287	74471
9999	-7	0	0	0	287	74462
10080	-7	0	0	0	287	74456
10089	-7	0	0	0	287	74454
10136	-7	0	0	0	287	74445
10254	-7	0	0	0	287	74359
10332	-7	0	0	0	287	74355
10445	-7	0	0	0	287	74354
10539	-7	0	0	0	287	74353
10761	-7	0	0	0	287	74352
10833	-7	0	0	0	287	74345
10862	-7	0	0	0	287	74332
11093	-7	0	0	0	287	74324
11572	-7	0	0	0	287	74323
11937	-7	0	0	0	287	74311
12083	-7	0	0	0	287	74309
12113	-7	0	0	0	287	74291
12308	-7	0	0	0	287	74284
12452	-7	0	0	0	287	74282
13081	-7	0	0	0	287	74281
13136	-7	0	0	0	287	74278
13477	-7	0	0	0	287	74277
14286	-7	0	0	0	287	74162
14598	-7	0	0	0	287	74160
14896	-7	0	0	0	287	74159
15300	-7	0	0	0	287	74145
16525	-7	0	0	0	287	74135
16815	-7	0	0	0	287	74118
17452	-7	0	0	0	287	74115
17891	-7	0	0	0	287	74114

17941	-7	0	0	0	287	74113
18887	-7	0	0	0	287	74111
20327	-7	0	0	0	287	74106
20364	-7	0	0	0	287	74101
20501	-7	0	0	0	287	74092
20502	-7	0	0	0	287	74003
21855	-7	0	0	0	287	73999
22033	-7	0	0	0	287	73981
22176	-7	0	0	0	287	73964
22341	-7	0	0	0	287	73882
23223	-7	0	0	0	287	73852
23583	-7	0	0	0	287	73840
23693	-7	0	0	0	287	73838
24459	-7	0	0	0	287	73837



A javítási lépések jobb megfigyelése érdekében az első 500 lépés egy részletesebb ábrán is bemutatásra kerül.

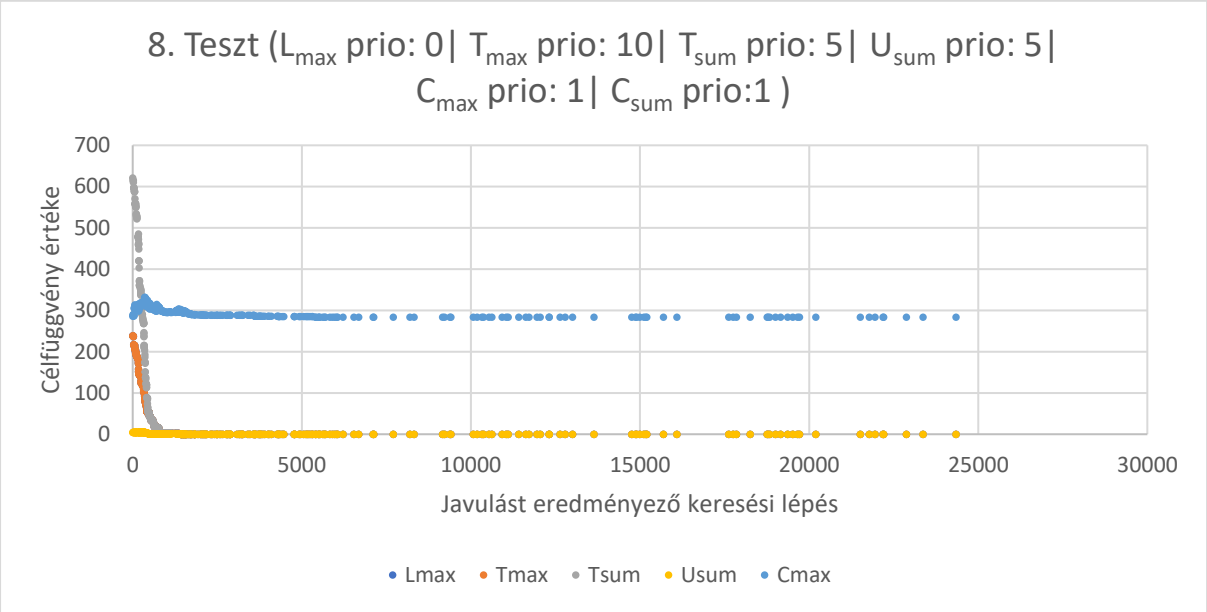


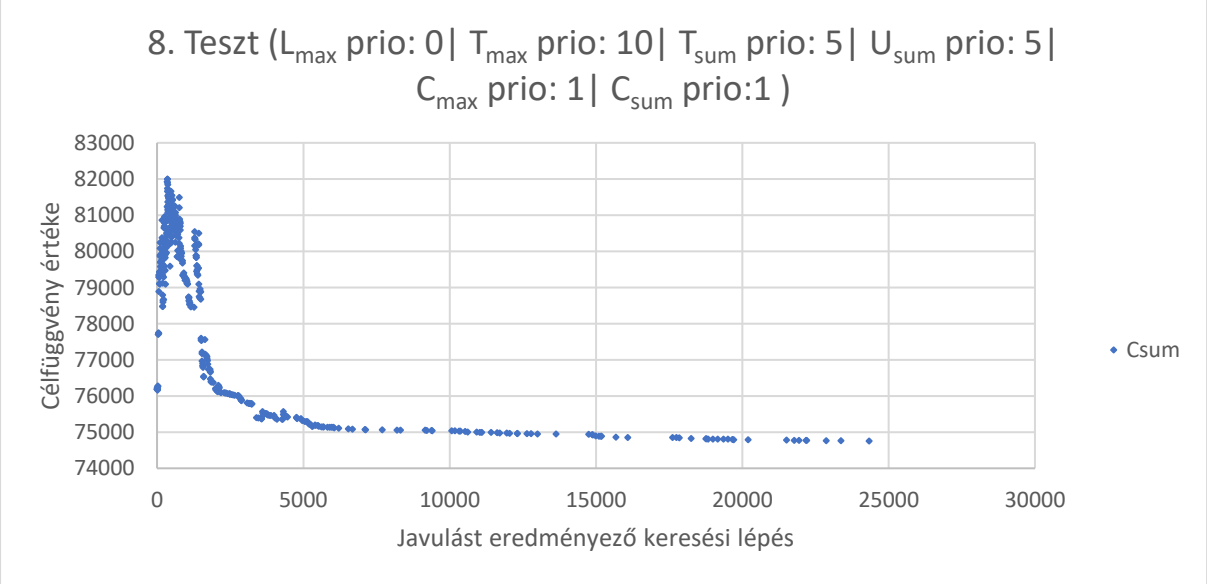
11.2.2 8. Teszt

A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	10	5	5	1	1

Terjedelmi okokból a 8. tesztet követően csak az eredményekből készített diagrammokat tartalmazza a függelék. A C_{sum} célfüggvény jelentősen eltérő értékkészlete a közös ábrák értelmezhetőségét rontja, ezért a célfüggvény értékeinek alakulása egy külön ábrán kerül bemutatásra.



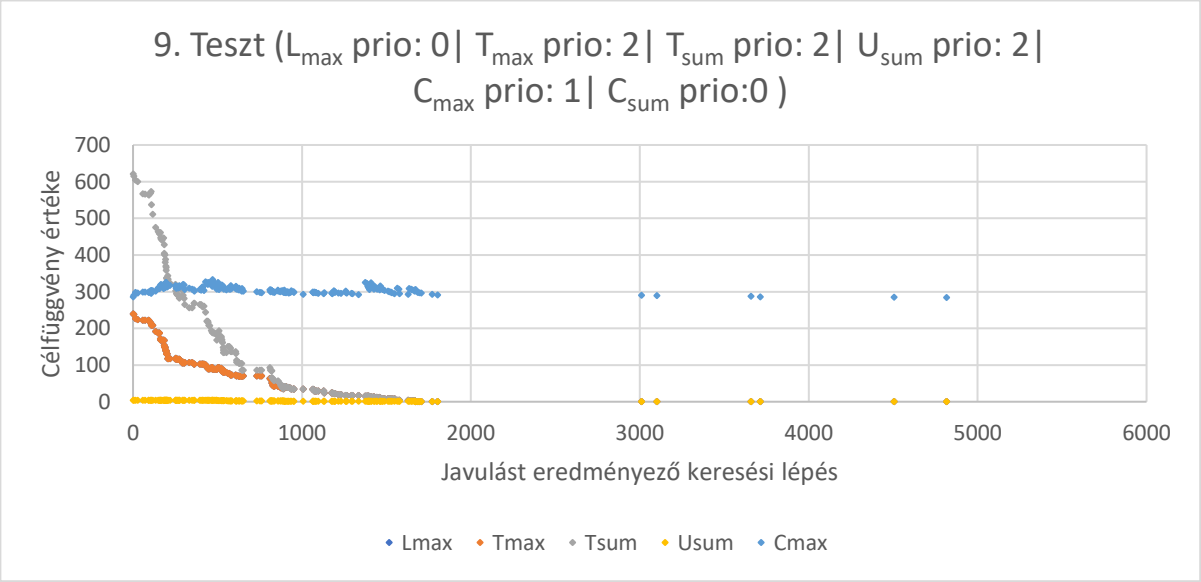


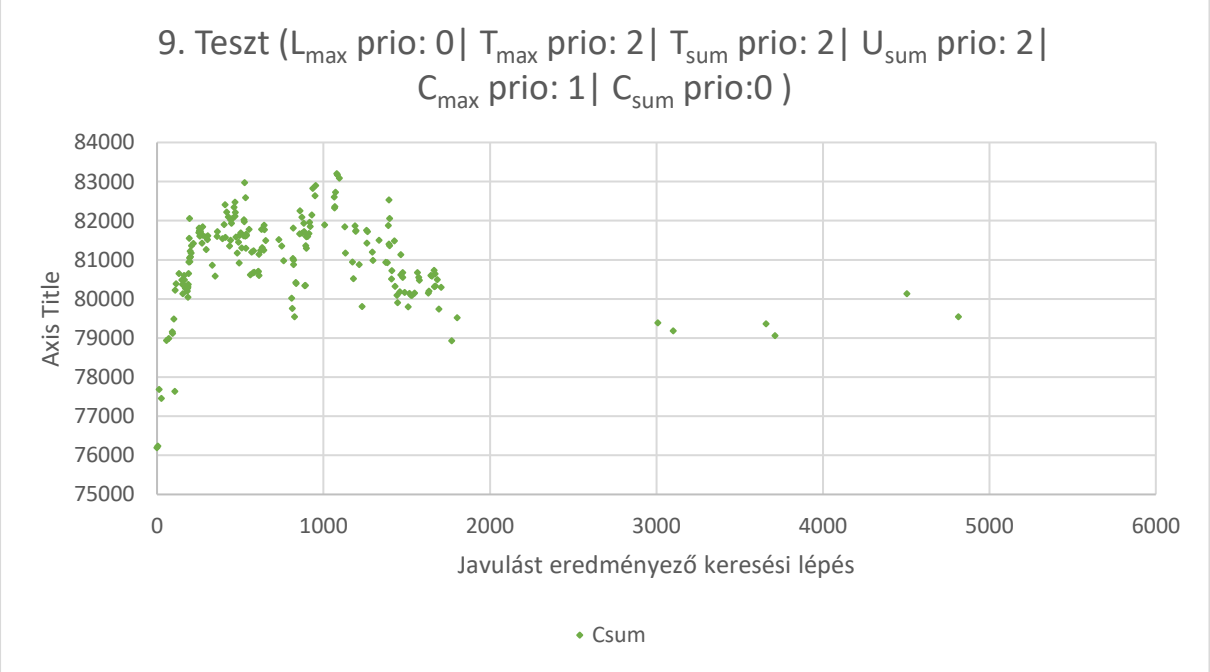
11.2.3 9. Teszt

A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	2	2	2	1	0

Terjedelmi okokból az eredményekből készített diagrammokat tartalmazza a függelék. A C_{sum} célfüggvény jelentősen eltérő értékkészlete a közös ábrák értelmezhetőségét rontja, ezért a célfüggvény értékeinek alakulása egy külön ábrán kerül bemutatásra.



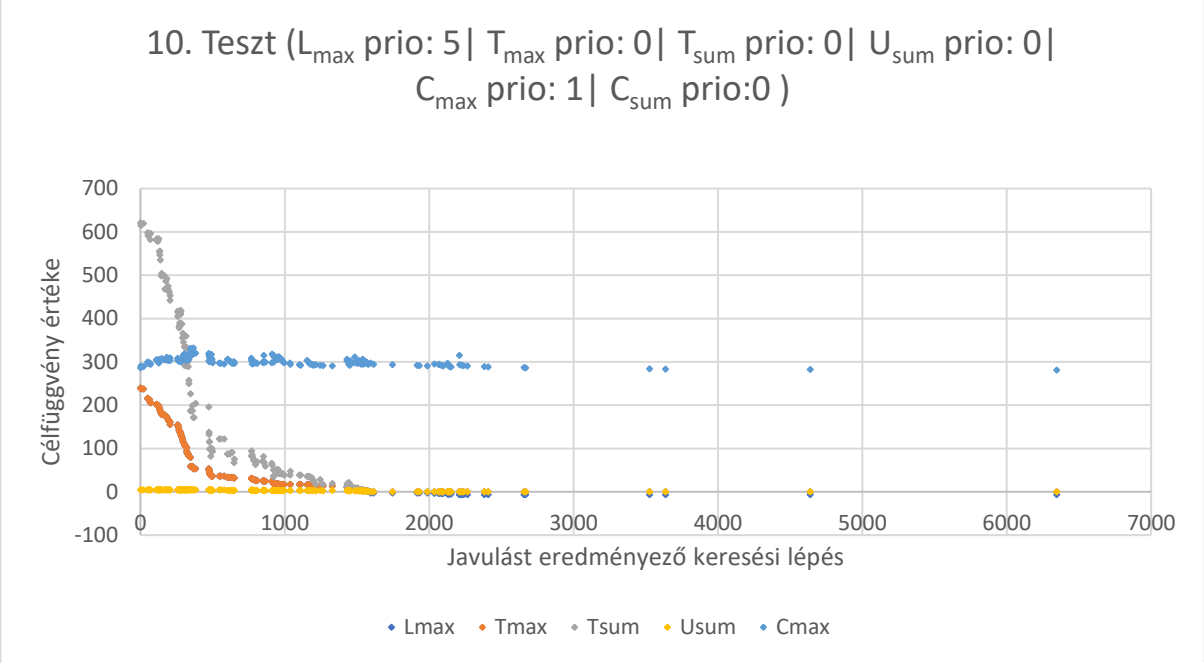


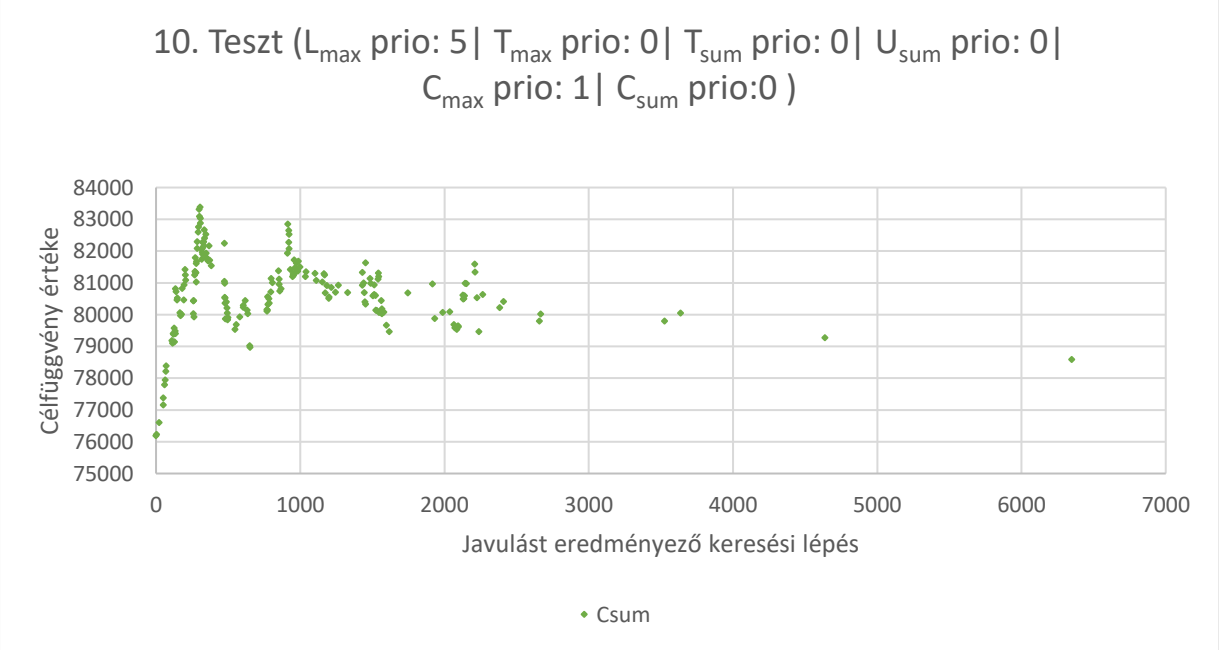
11.2.4 10. Teszt

A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
5	0	0	0	1	0

Terjedelmi okokból az eredményekből készített diagrammokat tartalmazza a függelék. A C_{sum} célfüggvény jelentősen eltérő értékkészlete a közös ábrák értelmezhetőségét rontja, ezért a célfüggvény értékeinek alakulása egy külön ábrán kerül bemutatásra.



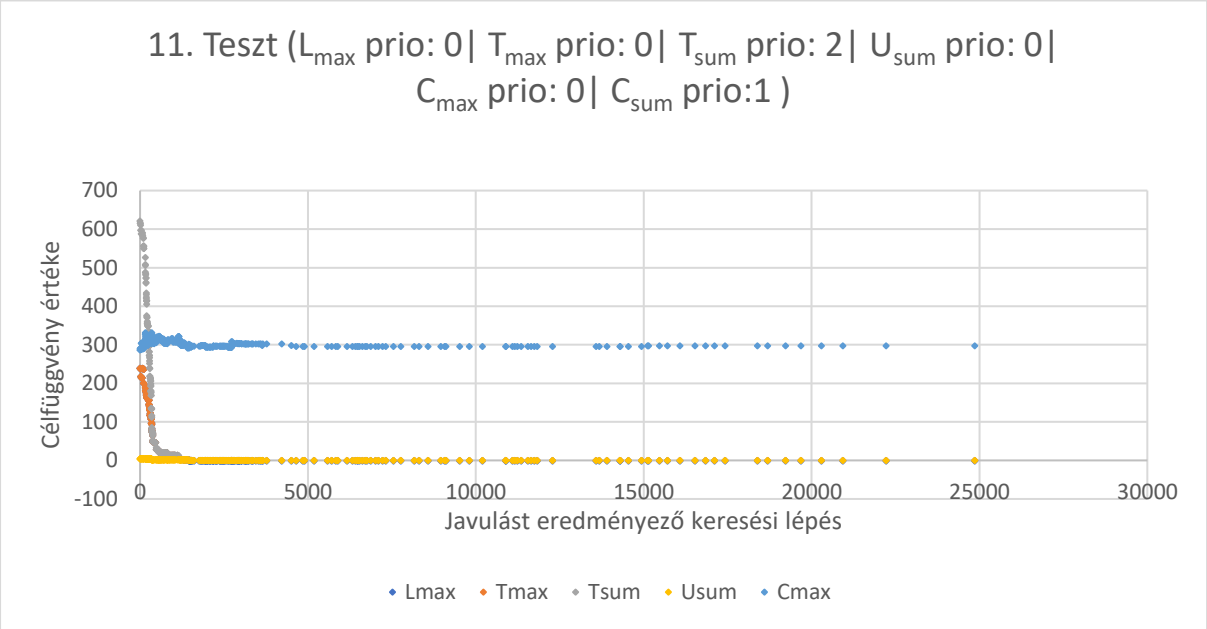


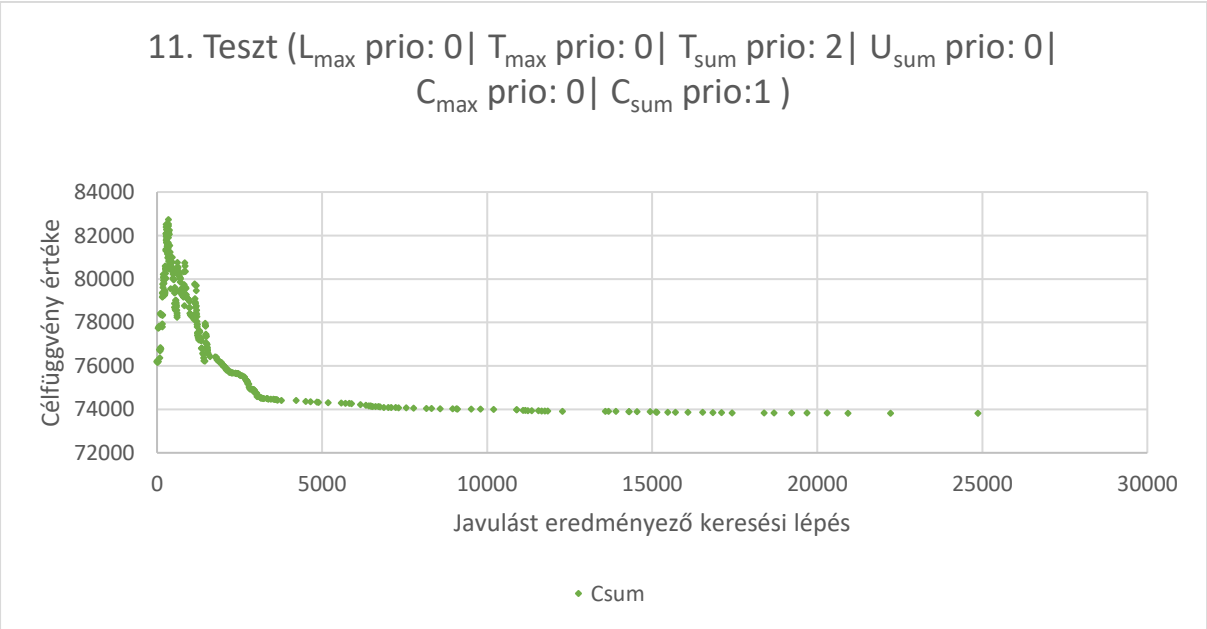
11.2.5 11. Teszt

A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	0	2	0	0	1

Terjedelmi okokból az eredményekből készített diagrammokat tartalmazza a függelék. A C_{sum} célfüggvény jelentősen eltérő értékkészlete a közös ábrák értelmezhetőségét rontja, ezért a célfüggvény értékeinek alakulása egy külön ábrán kerül bemutatásra.



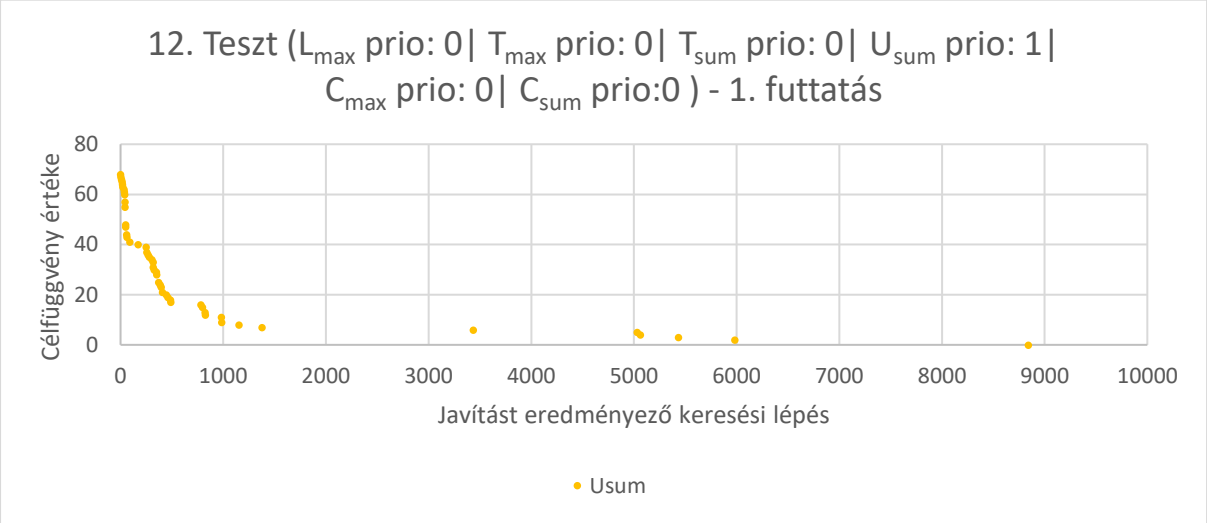


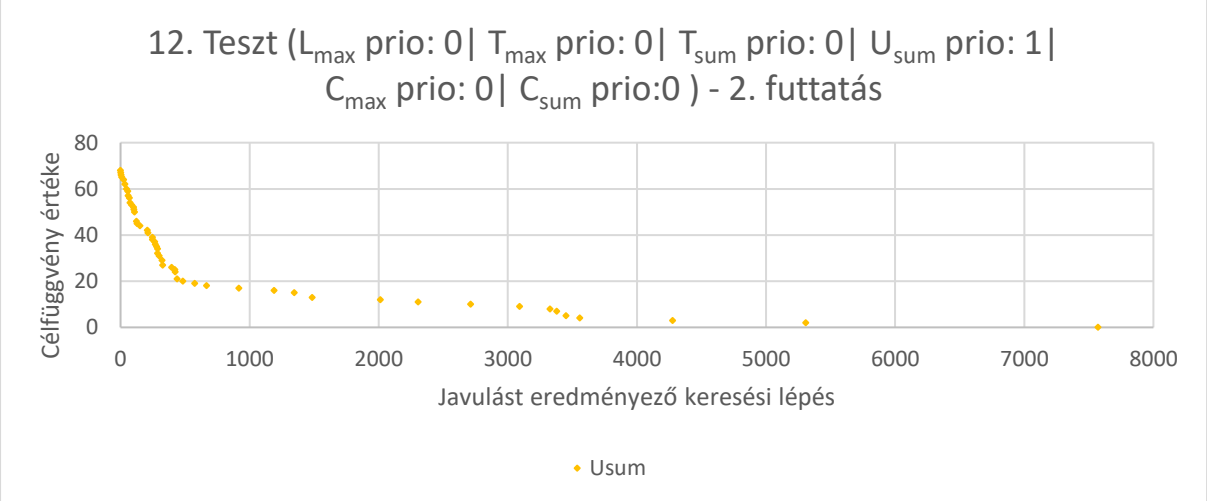
11.3 3. tesztcsoport

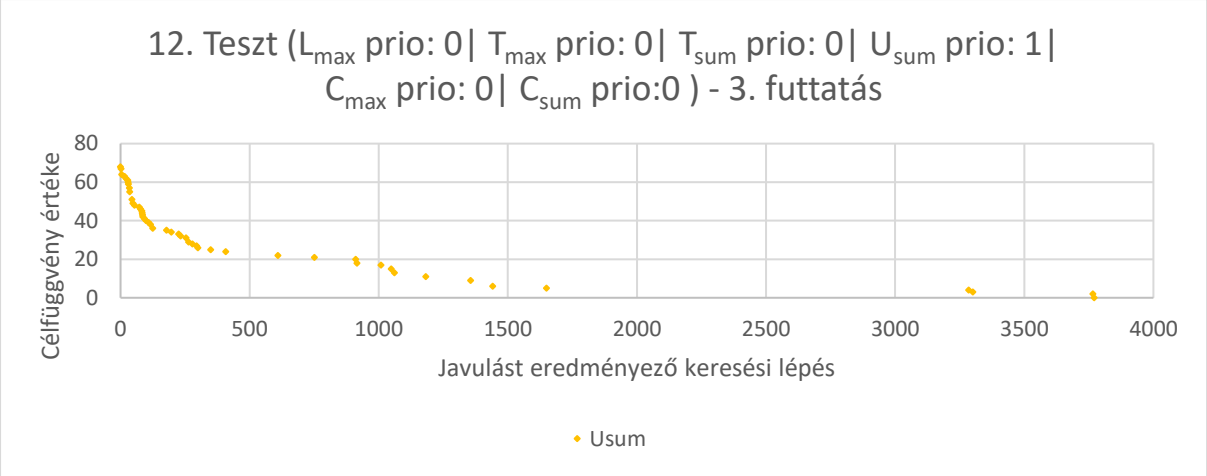
11.3.1 12. Teszt

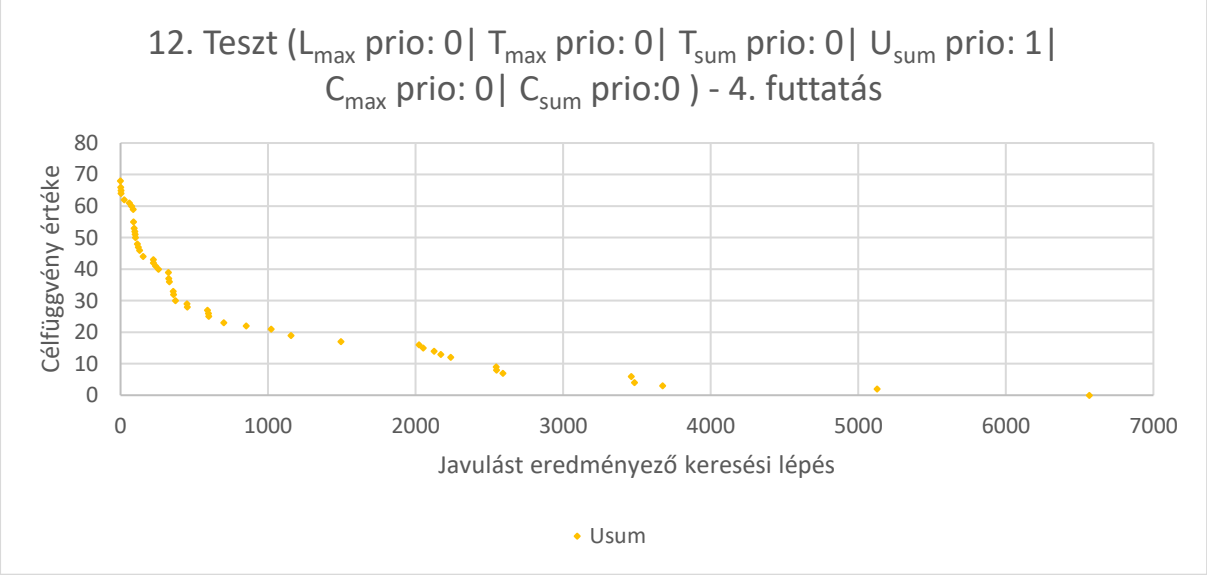
A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

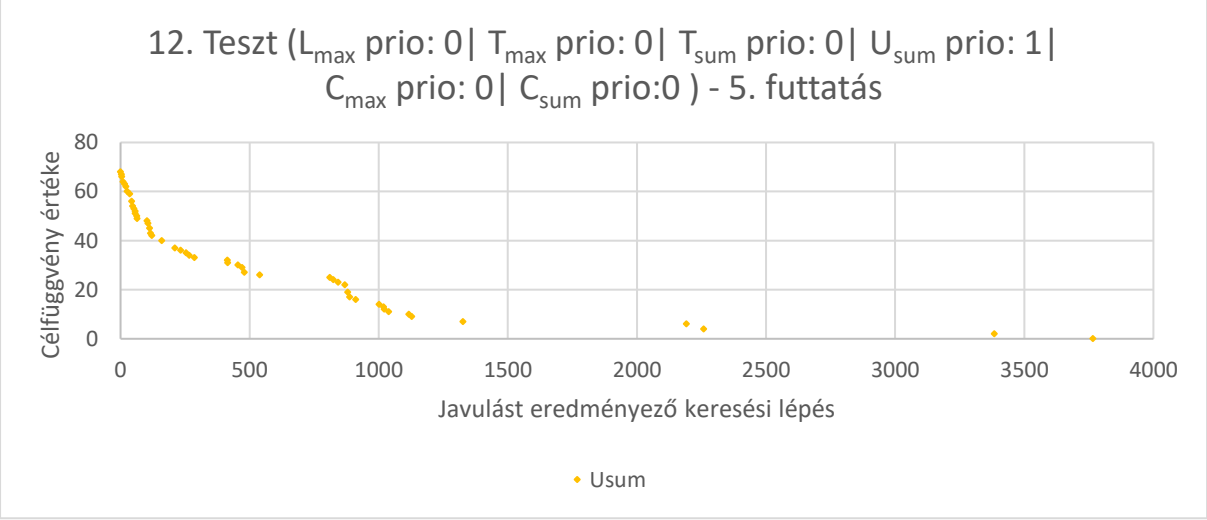
L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	0	0	1	0	0






12. Teszt (Lmax prio: 0| Tmax prio: 0| Tsum prio: 0| Usum prio: 1| Cmax prio: 0| Csum prio:0) - 3. futtatás
Célfüggvény értéke
80
60
40
20
0
0
500
1000
1500
2000
2500
3000
3500
4000
Javulást eredményező keresési lépés
Usum

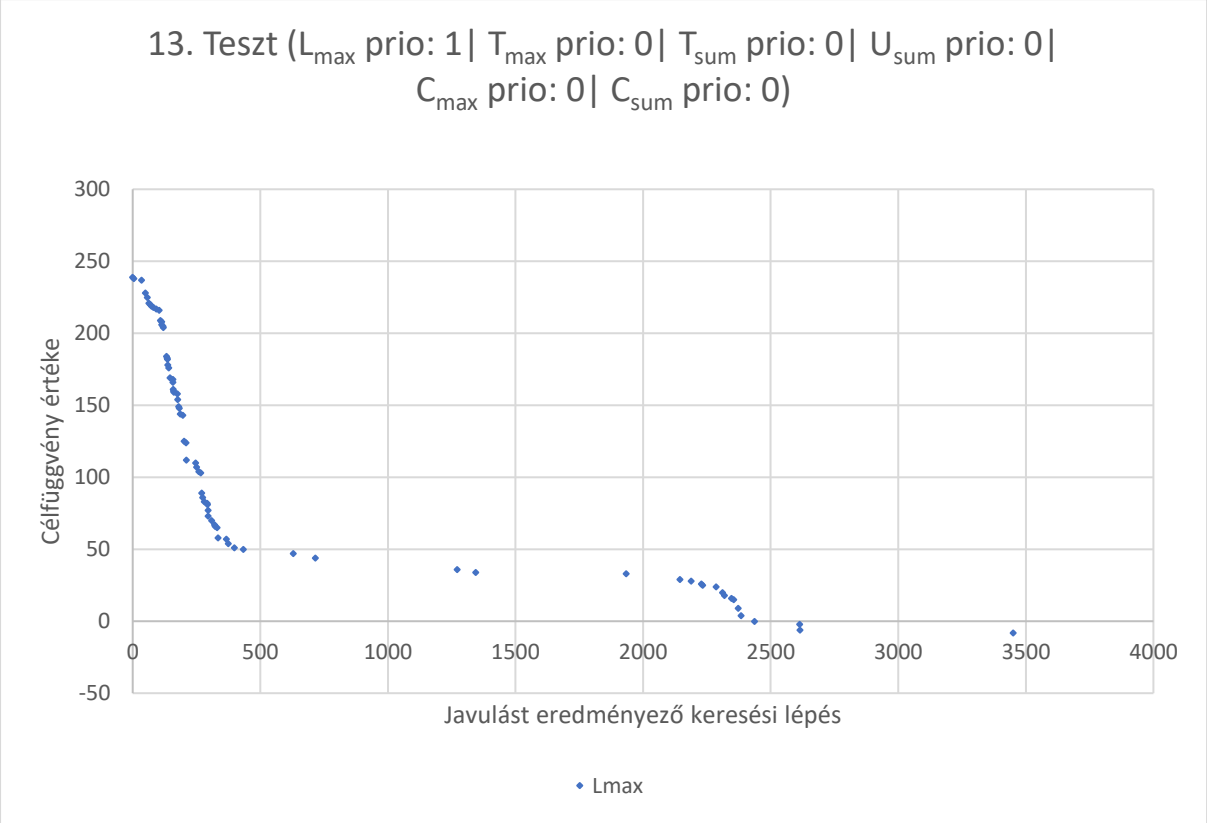

12. Teszt (Lmax prio: 0| Tmax prio: 0| Tsum prio: 0| Usum prio: 1| Cmax prio: 0| Csum prio:0) - 4. futtatás
Célfüggvény értéke
80
70
60
50
40
30
20
10
0
0
1000
2000
3000
4000
5000
6000
7000
Javulást eredményező keresési lépés
Usum


12. Teszt (Lmax prio: 0| Tmax prio: 0| Tsum prio: 0| Usum prio: 1| Cmax prio: 0| Csum prio:0) - 5. futtatás
Célfüggvény értéke
80
60
40
20
0
0
500
1000
1500
2000
2500
3000
3500
4000
Javulást eredményező keresési lépés
Usum

11.3.2 13. Teszt

A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

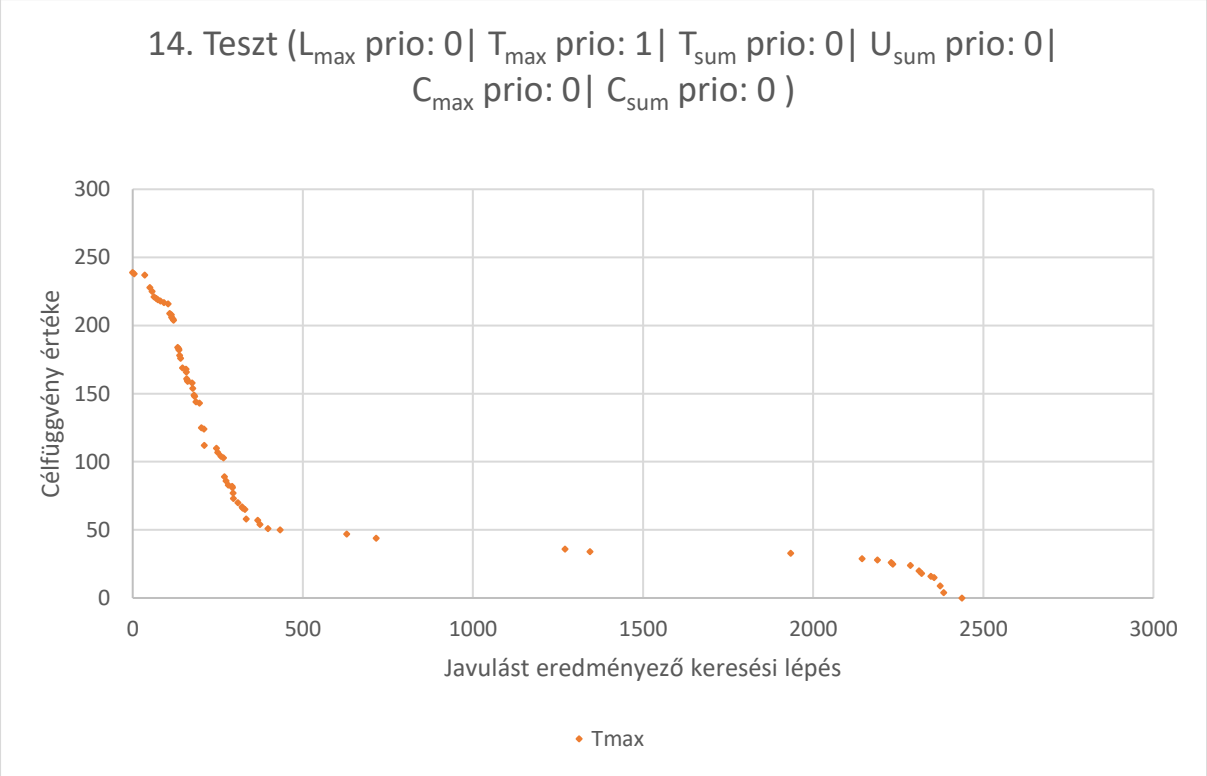
L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
1	0	0	0	0	0



11.3.3 14. Teszt

A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

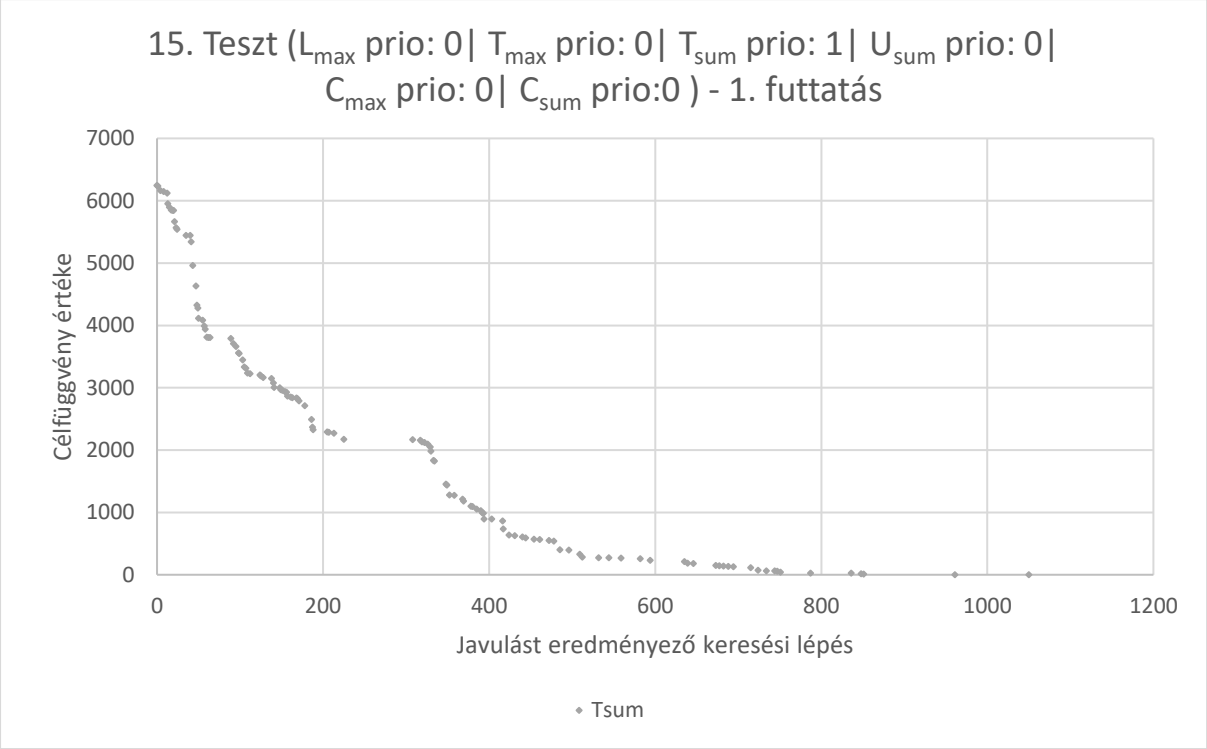
L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	1	0	0	0	0



11.3.4 15. Teszt

A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

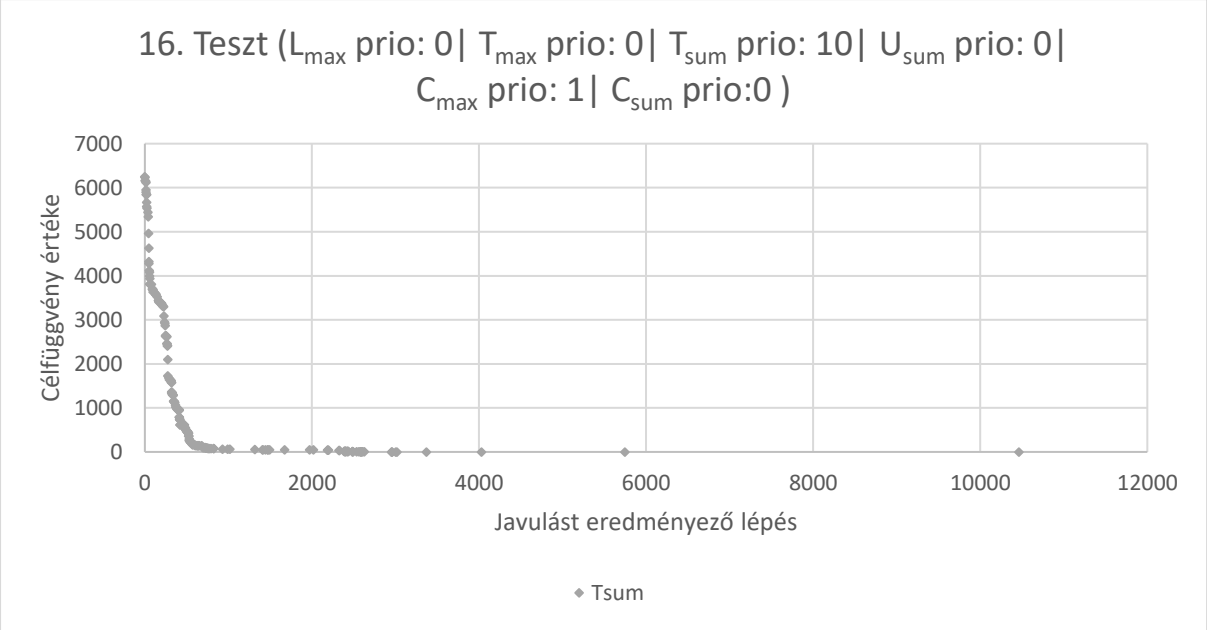
L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	0	1	0	0	0

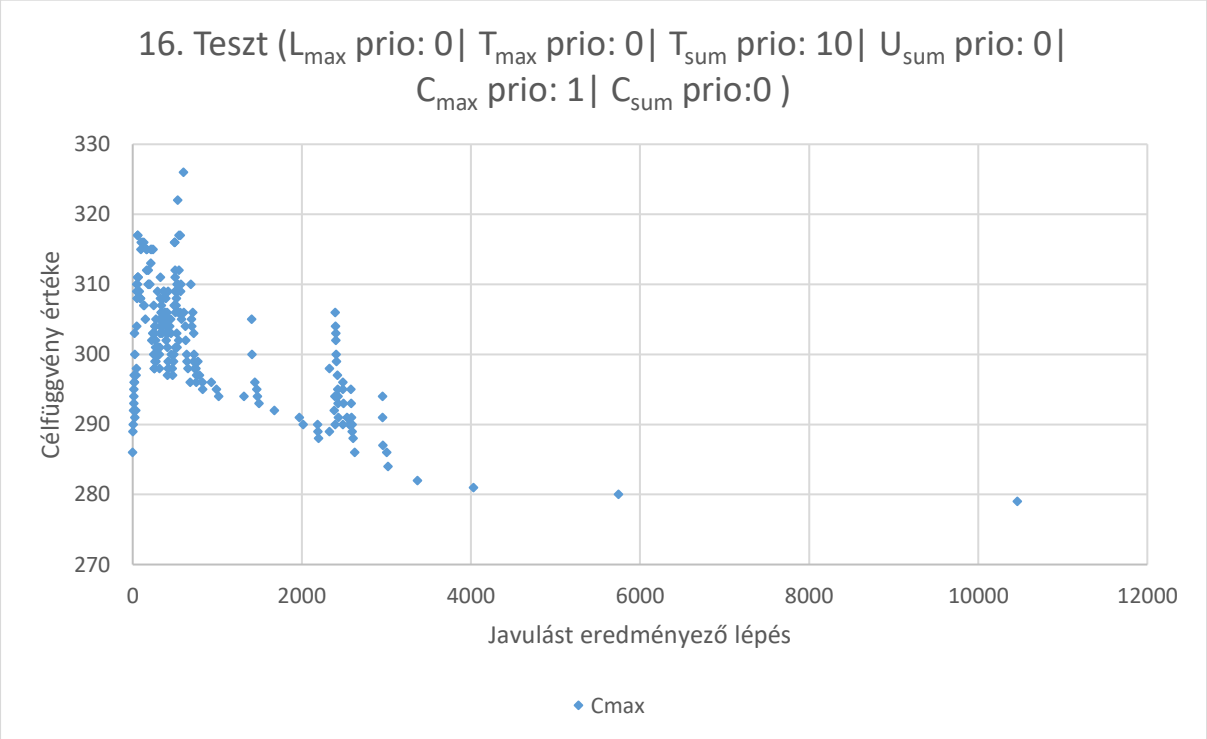


11.3.5 16. Teszt

A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	0	10	0	1	0

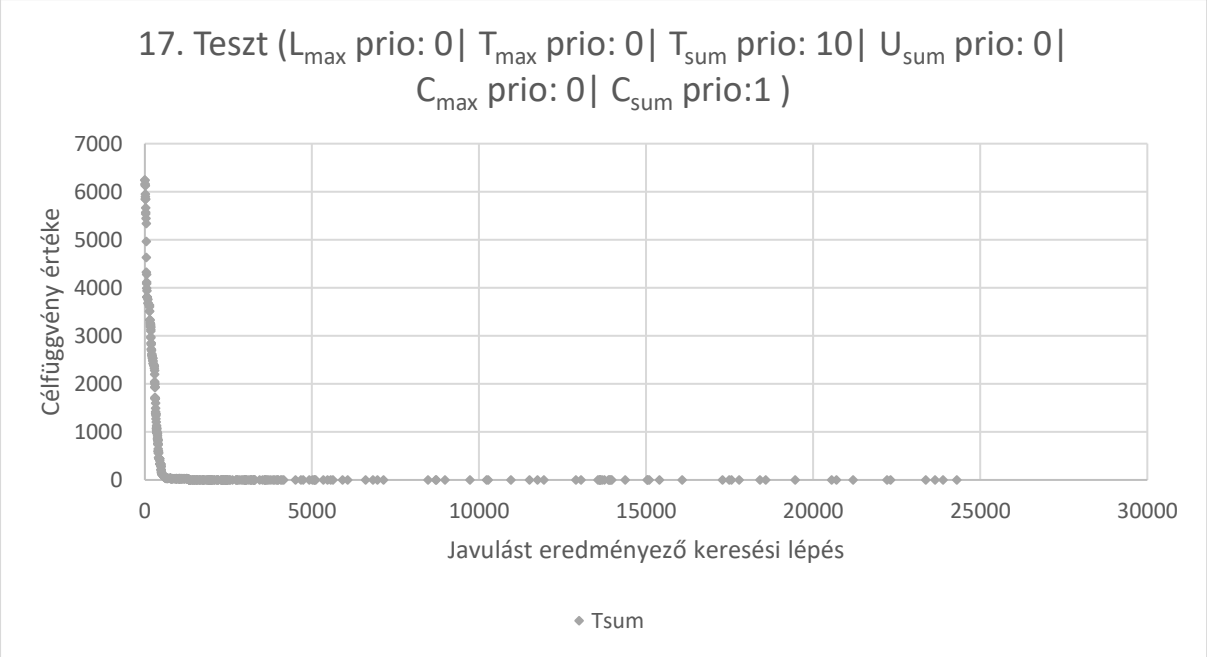


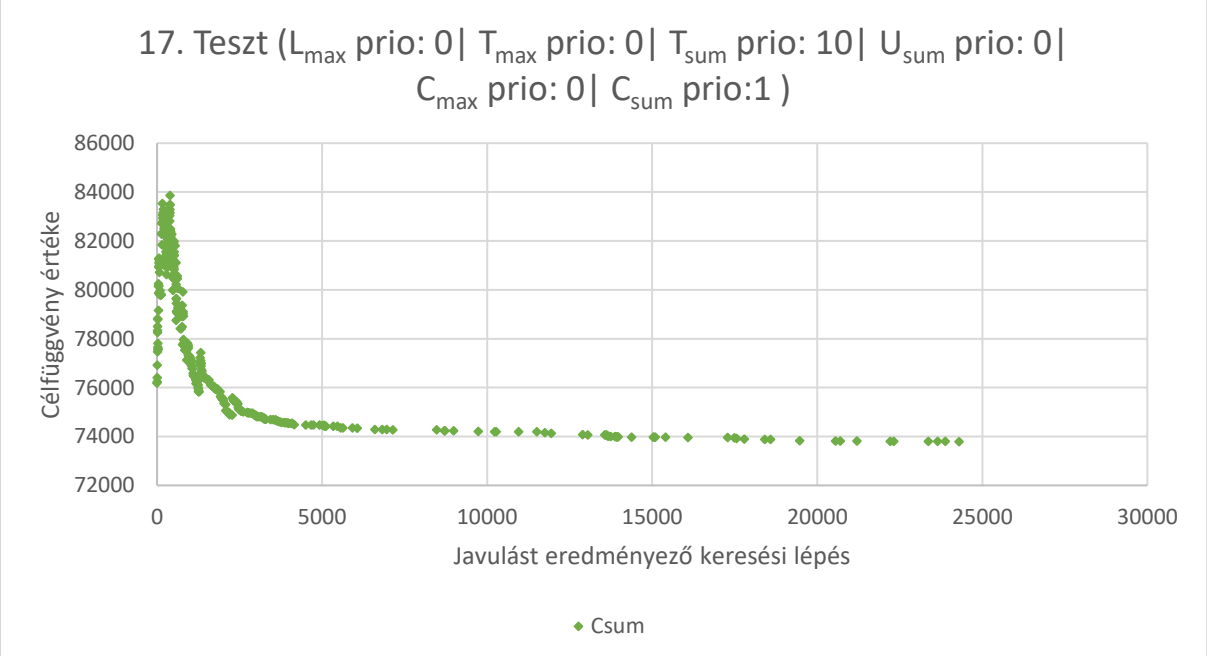


11.3.6 17. Teszt

A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	0	10	0	0	1



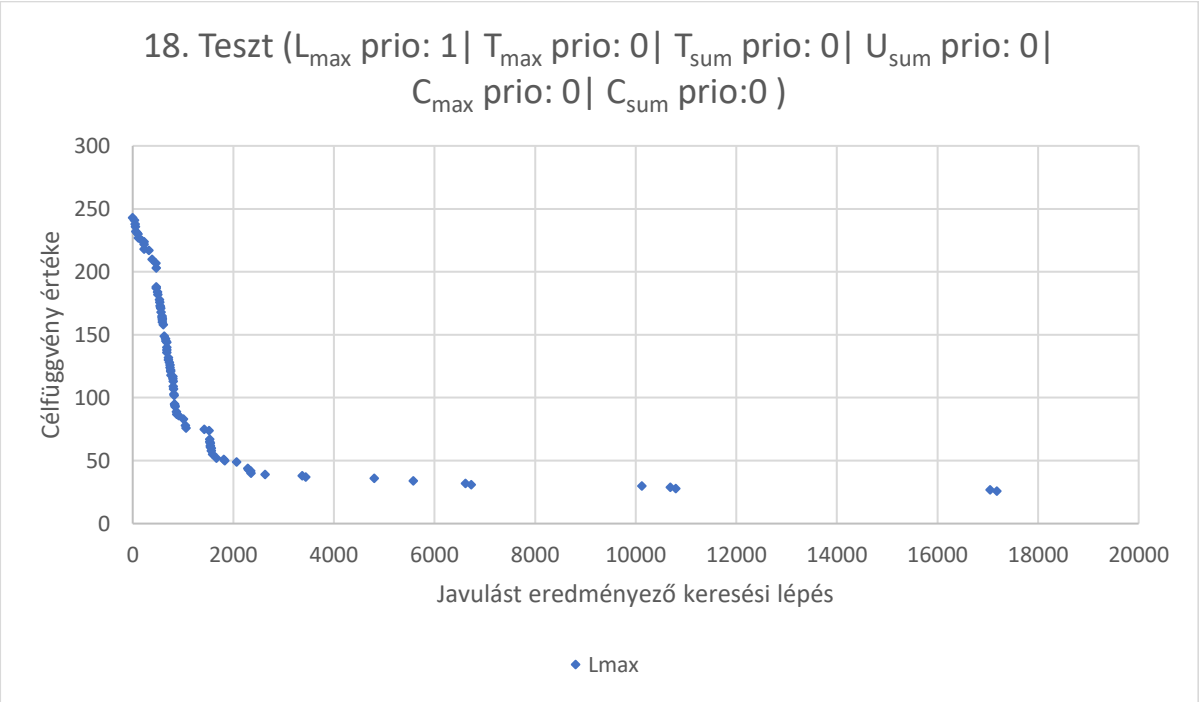


11.4 4. tesztcsoport

11.4.1 18. Teszt

A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

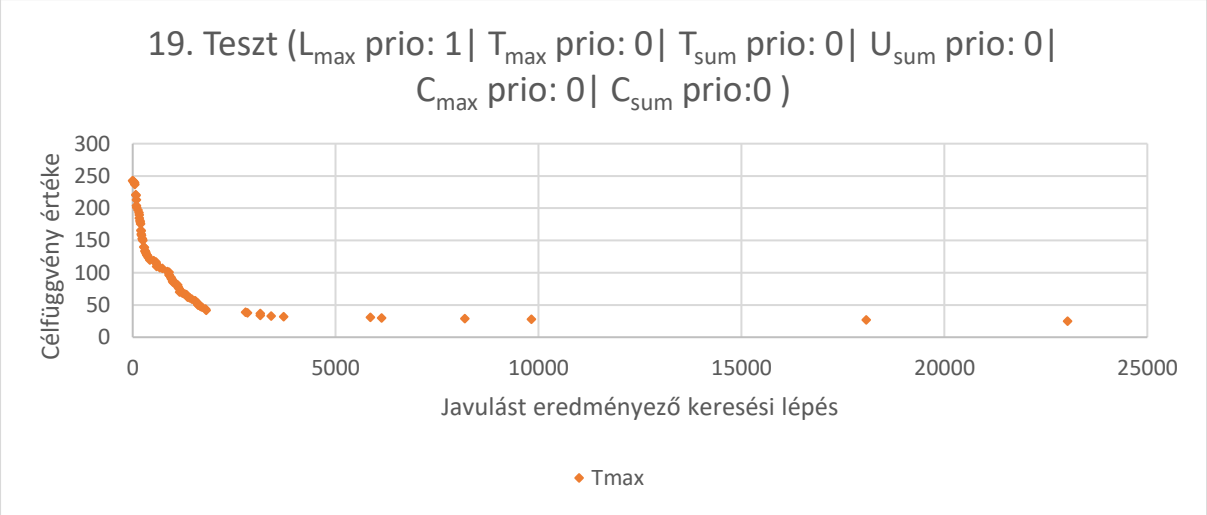
L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
1	0	0	0	0	0



11.4.2 19. Teszt

A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

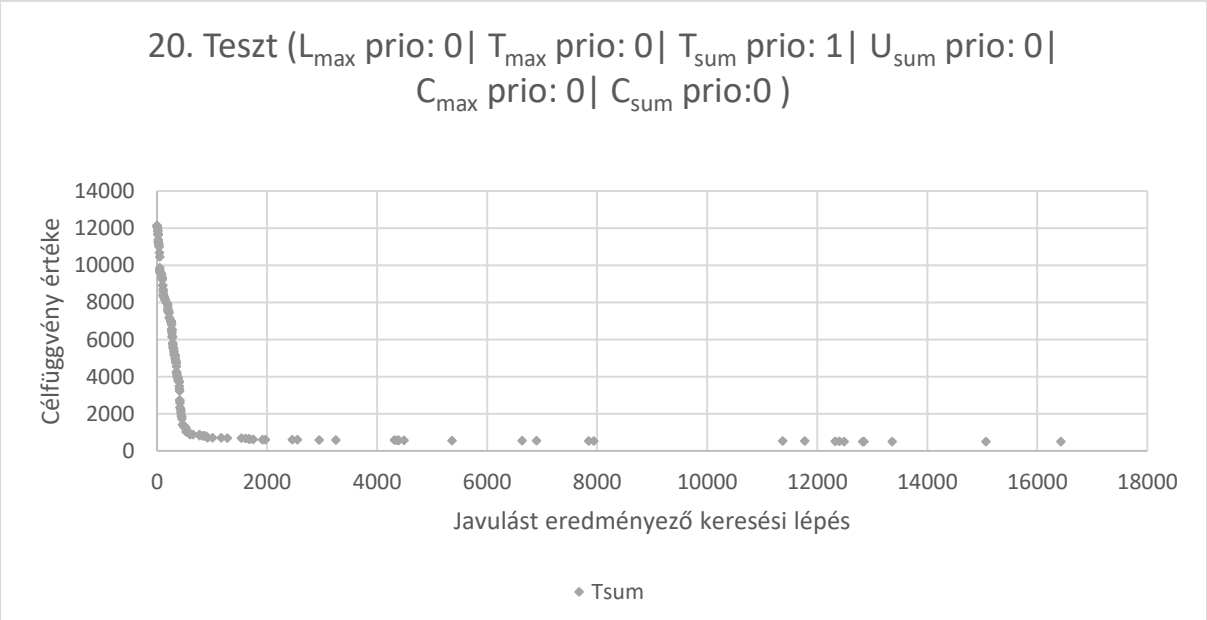
L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	1	0	0	0	0



11.4.3 20. Teszt

A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

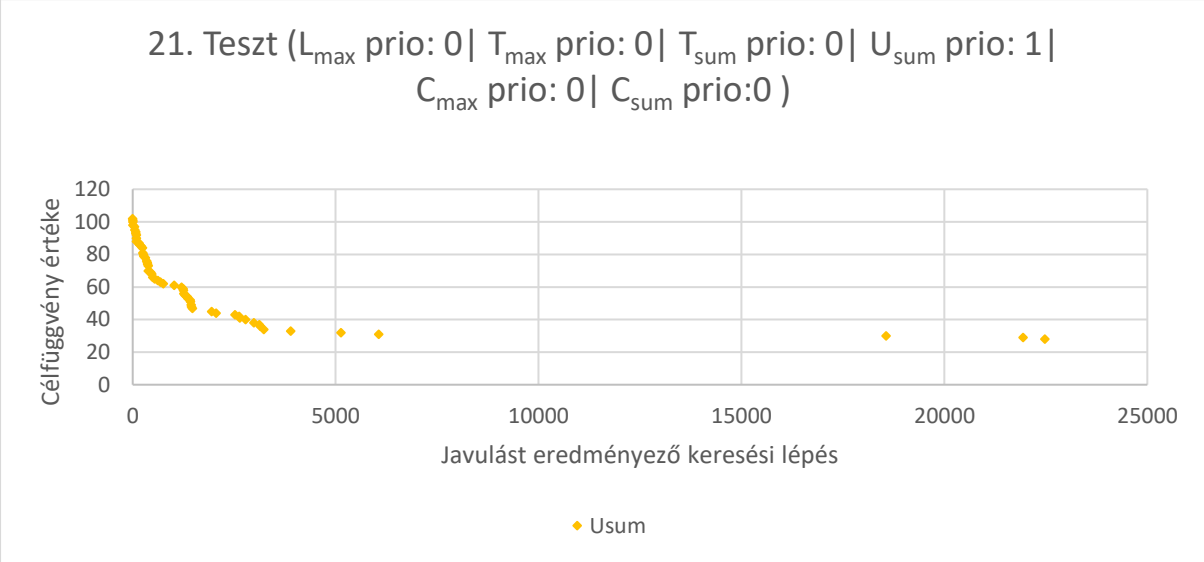
L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	0	1	0	0	0



11.4.4 21. Teszt

A teszt az alábbi célfüggvén-prioritásokkal került végrehajtásra:

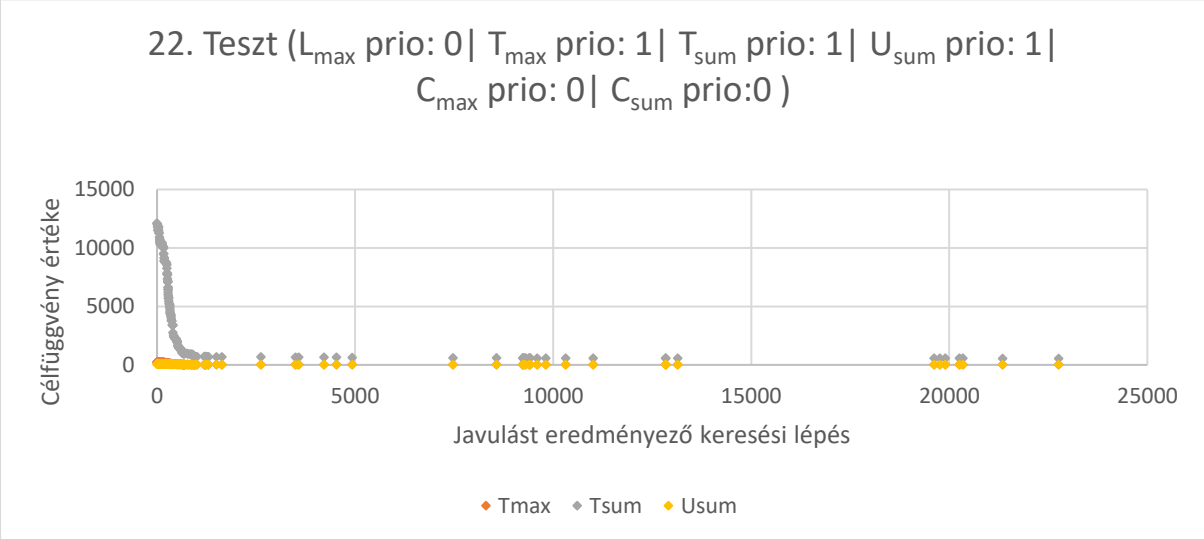
L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	0	0	1	0	0



11.4.5 22. Teszt

A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

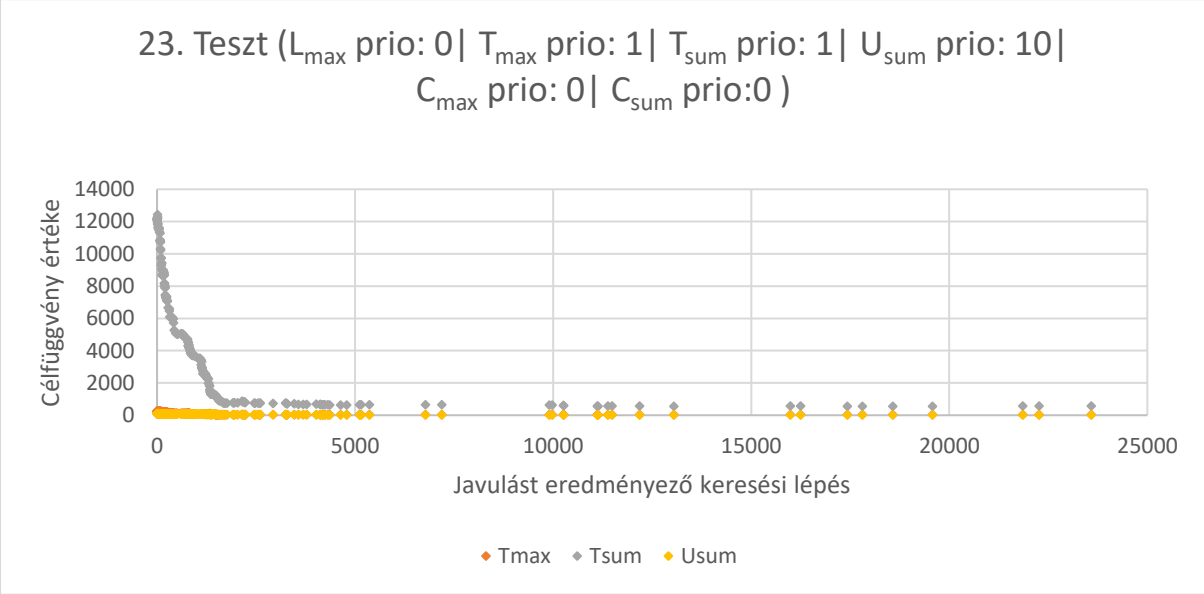
L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	1	1	1	0	0



11.4.6 23. Teszt

A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

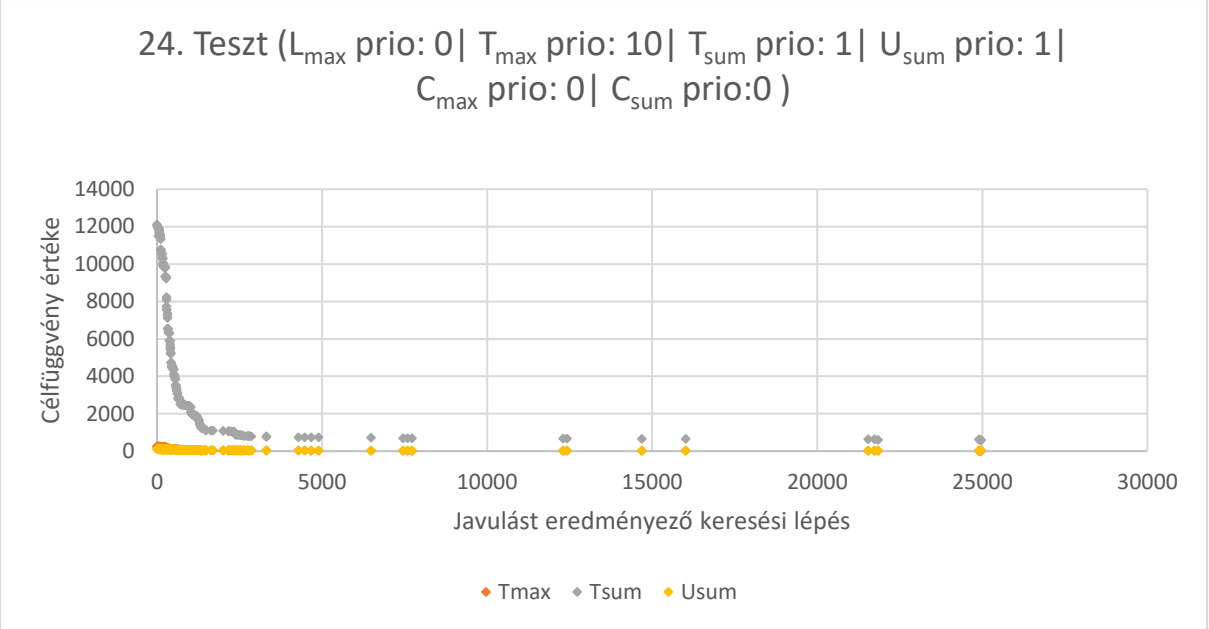
L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	1	1	10	0	0



11.4.7 24. Teszt

A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

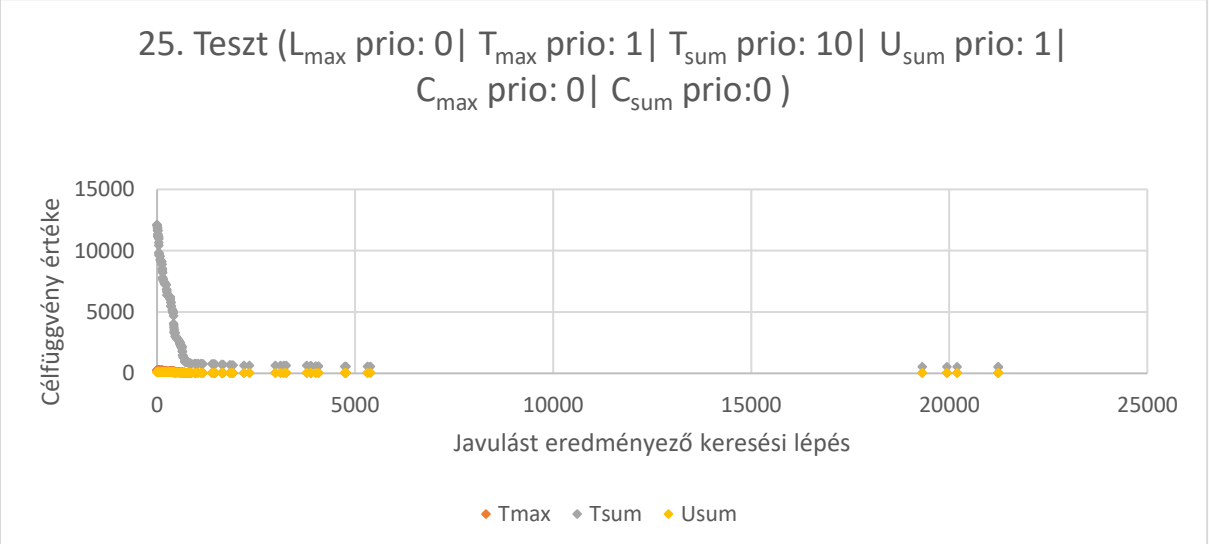
L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	10	1	1	0	0



11.4.8 25. Teszt

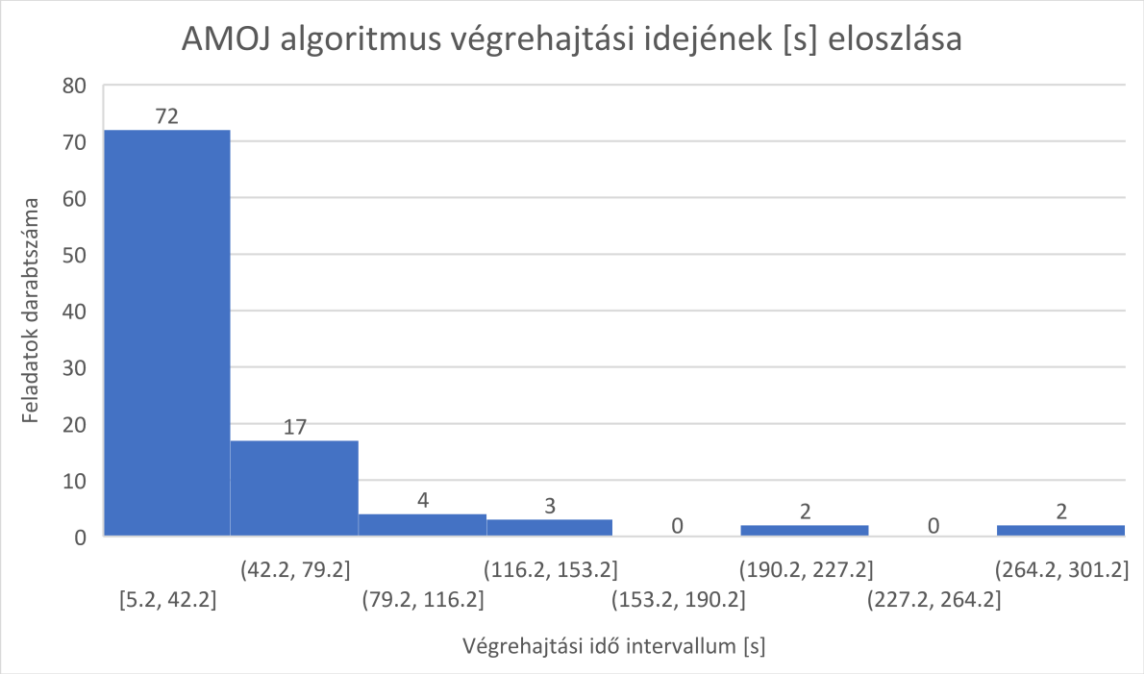
A teszt az alábbi célfüggvény-prioritásokkal került végrehajtásra:

L_{max}	T_{max}	T_{sum}	U_{sum}	C_{max}	C_{sum}
0	1	10	1	0	0



12 Függelék: Algoritmusok végrehajtási idejének mérései

12.1 AMOJ algoritmus végrehajtási ideje generált, 30 feladatot tartalmazó $1||C_{sum}$ feladatokon



12.2 MOSM algoritmus végrehajtási ideje RCPSP J30 benchmark feladatokon, maximálisan 50000 célfüggvény kiértékelési felső korláttal