Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Molnár Máté Lajos, Pintér Benedek

MUNKAIDŐ-BEOSZTÁS TERVEZÉSI ALGORITMUSOK ÖSSZEHASONLÍTÁSA

KONZULENS

Dr. Kővári Bence

BUDAPEST, 2019

Tartalomjegyzék

Összefoglaló

A kézzel történő munkaidő beosztás készítése egy rendkívül körülményes feladat a szerteágazó követelményrendszer miatt, ami ráadásul nem is ellentmondásmentes. Például sok dolgozó közel ugyanabban az időszakban szeretne nyaralni menni, miközben a munkáltatónak biztosítania kell az üzleti tevékenység folytonosságát. Továbbá a munkáltatók érdeke a foglalkoztatási költség minimalizálása. Az automatizált munkaidő tervezés célja az optimális beosztás megtalálása az adott követelmények mentén, miközben figyelembe veszi a munkajogi előírásokat. Sajnos ez egy NP-nehéz probléma, amit napjainkban is aktívan kutatnak.

Gyakori megközelítések közé tartoznak az egészértékű programozás és annak különféle vállfajai. Ez a megközelítés lineáris egyenletek segítségével fejezi ki a megkötéseket és a célfüggvényt, amire optimalizálni kell a feladatot, és előnye, hogy garantáltan megtalálja az optimális beosztást. Hátránya a memória éhsége és hosszú futási ideje.

Egy másik megközelítés a beosztás tervezésre a progresszív algoritmus, ami időben előre haladva készíti el a beosztást. A keresési tér nagyságrendekkel csökken ennél az algoritmus típusnál az időben előre haladó karakterisztika miatt. Ennek ára, hogy a követelmények kielégítése limitációkba ütközik.

E dokumentum részletesen taglalja az említett két megközelítést és összehasonlítja őket a Scheduling Benchmarks [1] nevű tudományos célokra létrehozott adathalmazon. Továbbá összehasonlítja a publikusan elérhető eddigi legoptimálisabb megoldásokkal is.

A dolgozat további lehetőségeket is taglal a fent említett két módszeren kívül. A progresszív módszer több módon is továbbfejleszthető, különös tekintettel az erőforrás szűkösség kezelésében. Az egészértékű programozási megvalósításban javulást eredményezhet az előzetes beosztások használata triviális esetekre. Továbbá a megerősítéses, illetve a mély tanulási algoritmusok is működő alternatívát nyújthatnak önállóan és hibrid megközelítésekben is.

Abstract

Employee scheduling by hand is a tedious and difficult task due to the wide variety of requirements, which often contradict each other. Many employees would like to go on holiday at roughly the same small period on summer, yet the employer must ensure the continuity of the business. Furthermore, they would like to minimize the cost of employment. Automated employee scheduling aims to tackle with the burden of labour laws and provide an optimal solution based on the additional requirements. Unfortunately, it proves to be an NP hard problem and still under active research.

Common approaches to the employee scheduling problem include various Integer Programming (IP) techniques. This family of approaches uses linear equations to describe constraints and an objective function to optimise for and has the ability to find the optimal solution. On the other hand, it faces multiple obstacles e.g. memory usage and execution time.

Another approach to solve employee scheduling is a progressive algorithm that produces the resulting schedule in a time forward manner. The search space can be reduced by orders of magnitude with this type of algorithm due to the time forward characteristic, at the additional cost of some limitations on requirements satisfaction.

This paper explores the two aforementioned approaches in greater detail and compares the results on a data set built for research purposes, called the Scheduling Benchmarks [1]. Through the publicly available data set, comparisons were made to the reference solutions as well.

For further experimentations reinforcement learning and deep learning algorithms seem as a viable option for the problem as a whole or for hybrid approaches. The progressive method also has rooms for improvement especially the handling of bottlenecks should be revised. The IP method could be improved with preliminary work scheduling for obvious cases.

1 Bevezetés

A munkaidő-beosztás elkészítése egy összetett feladat a munkahelyen, amely nagy tapasztalatot és szaktudást igényel a tervezőjétől. Gondoljunk csak bele, hogy legtöbben a saját foglalkoztatásukra vonatkozó szabályokkal sincsenek tisztában, ami a szerteágazó és időről időre változó munkatörvények tekintetében nem is olyan meglepő. Tovább bonyolítja a képet, hogy nem csak az adott hónapot kell figyelembe venni, mert vannak teljes évre, vagy bizonyos foglalkoztatási formákban akár több évre vonatkozó szabályok is, ezeket szintén mind észben kell tartani a tervezés során. Szerencsére léteznek már beosztás tervezést támogató szoftverek, amik segítenek a Munka törvénykönyvének (Mt.) [2] megfelelő beosztást készíteni, de ezek csupán jelezni tudják a tervezőnek, ha éppen valamilyen szabály nem teljesül [3]. Innen még egy óriási szakadék választ el minket a teljesen automatizált tervezéstől mind funkcionalitásban, mind számításelméleti kihívásban. De mik is azok az extra funkcionalitások, amiket jelenleg a tervezőnek kell manuálisan végrehajtania?

A Mt. valójában csak egy kötelezően betartandó minimális szabályrendszer a beosztástervezés során. A tervezőnek ezen felül figyelnie kell az egyenletes terheléselosztásra a dolgozók között, a szabadnap kérelmek kielégítésére, a munkafeladatok maradéktalan beosztására – ami munkaerőhiányos környezetben újabb fordulatot vesz –, vagy éppen a munkabér összköltségének minimalizálására. Ilyen, és ehhez hasonló jellegű követelményekben már nem tud segíteni egy döntéstámogató szoftver, és nagyobb létszámú munkahelyeken különös kihívást jelent a manuális tervezés során is.

Az imént említett követelmények már sejtetni engedik, hogy számításelméleti oldalról megközelítve az automatizációt, egy optimalizálási problémáról van szó. Játsszunk egy kicsit a számokkal! Tegyük fel, hogy 8 dolgozót tudunk beosztani és két héten keresztül, minden nap csupán egyetlen feladatot kell elvégezni. Első napra választhatom a 8 dolgozó bármelyikét, második napra is, és így tovább. Már ebben az egyszerű esetben is $8^{14} \cong 4$ ezer milliárd (!) lehetséges beosztás közül kell kiválasztania egy programnak az optimális beosztást. Ezzel a gyakorlatban nem tudnak megbirkózni a mai számítógépek, ezért különböző trükköket igyekeznek bevetni a terület kutatói. A terület meglehetősen gyerekcipőben jár még, nem ismert olyan beosztástervező szoftver,

ami a követelmények kellően széles skálájával meg tudna birkózni. Viszont vannak már megoldások egy-egy szűk területre [3], illetve a mesterséges intelligencia előretörése is nagy fordulatot rejthet magában az automatizált beosztástervezés területén.

Összefoglalva, a beosztástervezés automatizálása garantálja a Mt. betartását és különböző követelmények mentén igazságosabbá, optimálisabbá teheti az elkészült beosztást, amiből mind a munkavállalók, mind a munkáltató profitál.

A dolgozat egy irodalmi áttekintéssel folytatódik, amiben sorra vesszük a beosztástervezéshez használt népszerű algoritmusokat, majd bemutatjuk a saját beosztástervezőinkhez használt teszt adathalmaz felépítését, szabályait, legjobb ismert eredményeit. A 3. és 4. fejezetben részletesen ismertetjük a munkánk során elkészített két algoritmus működését, az 5. fejezetben összehasonlítjuk őket a teszt adathalmazon. Végül röviden összefoglaljuk a dolgozatot és kitekintünk a továbbfejlesztési lehetőségekre.

2 Irodalmi áttekintés

A munkaidő-beosztás tervezés problémája leírható úgy, hogy dolgozókat rendelünk feladatokhoz olyan módon, hogy minden egyes feladathoz legfeljebb egyetlen dolgozó van hozzárendelve [4]. Ennél részletesebben megfogalmazva a problémát már különböző szabályokat vonunk be a definícióba. Például igaz-e a dolgozó-feladat kapcsolatra, hogy 1-1 számosságú, vagy lehet akár 1-több számosságú is? Sok munkahelyen kötelező, hogy egy egészségügyi ellátás képesítéssel rendelkező személy is jelen legyen a munkavégzés ideje alatt, de eközben ő végezhet más feladatot is, azaz két, párhuzamosan végezhető feladathoz rendeltük.

Az imént említett munkahelyi sajátosság csak egy volt a sok közül, ezen felül rengeteg olyan követelmény merülhet fel, ami nem általános érvényű, ezeket [5] részletesen összefoglalja. A követelményrendszert tovább bonyolítja, hogy országonként akár jelentős eltérések is lehetnek a törvényi szabályozásokban. Jelenleg a követelményrendszer is egy feltáratlan kutatási terület, mivel nincs a gyakorlatban elterjedt automatikus beosztástervező szoftver, és nincs átható, gyakorlati tapasztalatokból származó képünk arról, hogy milyen igények merülnek fel felhasználói oldalról.

A műszak szerinti beosztástervezés sokat könnyít az automatizációban, ezért a kutatási célú algoritmusok területén felkapott téma az egészségügyben dolgozó ápolók beosztásának tervezése, ahol 0-24-es ellátásra van szükség és tipikusan 3 műszakban dolgoznak. Erre ír fel egy modellt és használ súlyozott SAT solver algoritmust [6], illetve hibrid IP algoritmust [7]. A gyakorlatban rengeteg olyan munkahely van, ahol nem műszakok szerint történik a beosztás, hanem egy sokkal finomabb felbontású, akár néhány perces időszeletekkel megtervezett beosztásra van szükség. Az időszelet alapú beosztástervezést használja [8] farmakológia kutatók és gyógyszer tesztelők beosztására.

A terület népszerűbb algoritmusai közül néhány [5]:

- Branch & Bound,
- Integer Programming,
- Genetikus algoritmusok,
- Metaheurisztikus algoritmusok.

Ezek mellett találkozhatunk egyéb, kevésbé intenzíven kutatott algoritmus típusokkal is, mint például az időben előre haladó módszer, amit mi ígéretesnek találtunk és megvalósítottuk, erről szól a 3. fejezet. Reptéri alkalmazottak beosztásához használja [9] az időben előre haladó algoritmust, majd tovább javítja az így elkezdett beosztást egy konstruktív/destruktív algoritmussal. A cikk megemlíti, hogy gyakorlati alkalmazással is kísérleteztek.

Laboratóriumi alkalmazottak beosztását készíti el [10], ami szintén több lépésben állítja össze a végleges beosztást és az időben előre haladó algoritmus végrehajtása után mohó módon javítja tovább az eredményt.

Az időben előre haladó módszer gráfokban keres párosítást a beosztás elkészítéséhez. Egy korábbi cikkünkben [11] összehasonlítunk különböző párosítást kereső algoritmusokat. [12]-ben pedig összehasonlítjuk, hogy milyen hatással van az elkészült beosztások minőségét jelző pontszámra, ha több körben, különböző szempontok alapján tervezzük meg a beosztást.

Az egészértékű programozás, innentől IP (Integer Programming) megközelítés gyakran használt az ütemezési és beosztás tervezési feladatok esetén. Ennek az oka, hogy az említett feladat típusok felírhatók egy hozzárendelési problémaként, amiben egy időponthoz vagy időszakhoz kerül hozzárendelésre egy probléma specifikus objektummal, esetünkben ez egy dolgozó-tevékenység páros. Egy ilyen hozzárendelési feladat modellezhető egy súlyozott teljes páros gráffal, amelynek a két csúcsosztálya az időszakok és a probléma specifikus objektum. A gráf élei egy időszak és egy objektum kapcsolatát írják le. Az élek súlyozása mondja meg, hogy hozzárendelés történt-e (pl.: az élsúly 0-s értéke modellezze azt, hogy az élhez tartozó objektum nincs hozzárendelve az élhez tartozó időszakhoz). Ez a modell a lehetővé teszi, hogy a gráf illeszkedési mátrixa további feltételek nélkül megfelel egy egészértékű programozási feladat együtthatómátrixával. Amennyiben konkrét ütemezési feladatra vonatkozó feltételek lineárisak és csak a hozzárendelésekre vonatkoznak a független változók száma nem változik további feltételek felvételével, amelyeket a meglévő változók felhasználásával új kényszerként fel.

Ez a megközelítés egy általános sémát ad, ugyanakkor, ha ez nem lenne elég kiterjesztett változatait különféle ütemezési problémák megoldására használják (MIP [13], MINLP [14]) éppen ezért napjainkig kutatott téma az NP-nehéz problémákat minél hatékonyabban megoldó Solverek (megoldóprogramok)

2.1 Scheduling Benchmarks

A Scheduling Benchmarks [15] egy publikusan elérhető adathalmaz, aminek segítségével összemérhetjük az algoritmusunkat más algoritmusokkal. Az adathalmaz 24 tesztesetet definiál, és a feladatunk, hogy az egészségügyben dolgozó ápolók számára tervezzünk beosztást a megadott szabályok alapján. A tesztesetek a 2014-es Nemzetközi Beosztástervezési Versenyen debütáltak [16].

A tesztesetek segítségével széles skálán tesztelhetjük algoritmusunkat, a legegyszerűbb esetben 2 hétre, 1 műszak típusra kell 8 dolgozót beosztani, míg a legbonyolultabb esetben már 1 éves időtartam, 32 műszak típus és 150 dolgozó szerepel. A feladatok nehézségét jól mutatja, hogy a legbonyolultabbra még nem sikerült az elméleti legjobb megoldást megtalálni, 7 esetre pedig ugyan megvan az elméleti, de még nem sikerült a gyakorlatban is elérni.

Az adathalmazban definiált szabályok mindegyik tesztesetnél ugyanazok, ezeket kemény és puha követelményekbe sorolhatjuk. A kemény követelmények megszegése esetén nem elfogadható egy megoldás, tehát ezeknek mindenképp teljesülnie kell, míg a puha követelmények esetében egy szélesebb skálán mozog az értékelés, ezeket nem kötelező betartani. Azonban a nem teljesülésük esetén büntetőpont jár az elkészült beosztásra, amik alapján meg lehet végül állapítani, hogy mennyire optimális az eredmény. Következzenek a szabályok [15]:

Kemény követelmények:

- A dolgozók nem oszthatók be több műszakra egy napon.
- Maximális műszakok száma: minden egyes műszaktípushoz meg van határozva, hogy maximum hány darab lehet belőle beosztva a dolgozóhoz.
- Minimális és maximális összmunkaidő: A dolgozó munkával tölthető perceinek minimális és maximális száma a teljes tervezési időszak alatt.
- Minimális pihenőidő: A dolgozó két műszakja között eltelt pihenőidő minimális hossza percben.
- Minimális és maximális egymást követő munkanapok száma: A dolgozó egyhuzamban dolgozott napjainak minimális és maximális száma.

- Minimális egymást követő pihenőnapok száma: A dolgozó egyhuzamban pihent napjainak minimális száma.
- Munkával töltött hétvégék maximális száma: A dolgozó munkával töltött hétvégéinek maximális száma. Egy hétvége munkával telt, ha szombaton, vagy vasárnap, vagy mindkét nap volt beosztása a dolgozónak.
- Pihenőnapok: A dolgozó nem osztható be megadott napokon.

Puha követelmények:

- Műszak preferenciák: egy adott nap adott műszaktípusára kérheti a dolgozó, hogy beosszák, vagy épp ellenkezőleg, ne osszák be. Ha nem kapja meg a beosztást, akkor a kérelemben meghatározott mennyiségű büntetőpont jár.
- Műszak igények: meg van határozva, hogy melyik nap hány dolgozóra van szükség egy adott műszakból. Ha ettől eltér a beosztás, akkor az igényhez meghatározott mennyiségű büntetőpont jár annyiszor, amennyi az eltérés az igényszámhoz képest. Kevesebb beosztásért több büntetőpont jár, mint a túlzott beosztásért.

Mindegyik teszteset úgy lett megalkotva, hogy kihívás legyen megoldani. A gyakorlatban ez azt jelenti, hogy nem lehet olyan beosztást tervezni, ami minden követelménynek eleget tesz, valamilyen kompromisszumot mindenképp kötni kell. Továbbá a tesztesetek nagyrésze erőforráshiányos is, vagyis kevesebb percet dolgozhat az összes dolgozó együttvéve a teljes tervezési időszak alatt, mint amennyi műszakra igény van.

Az alábbi táblázat összefoglalja a tesztesetek bonyolultságát, illetve az eddig ismert legjobb megoldásokat, és a legjobb automatikus beosztástervezéssel elért megoldásokat. A vastagon szedett pontszámok bizonyítottan a legjobb elérhető megoldások.

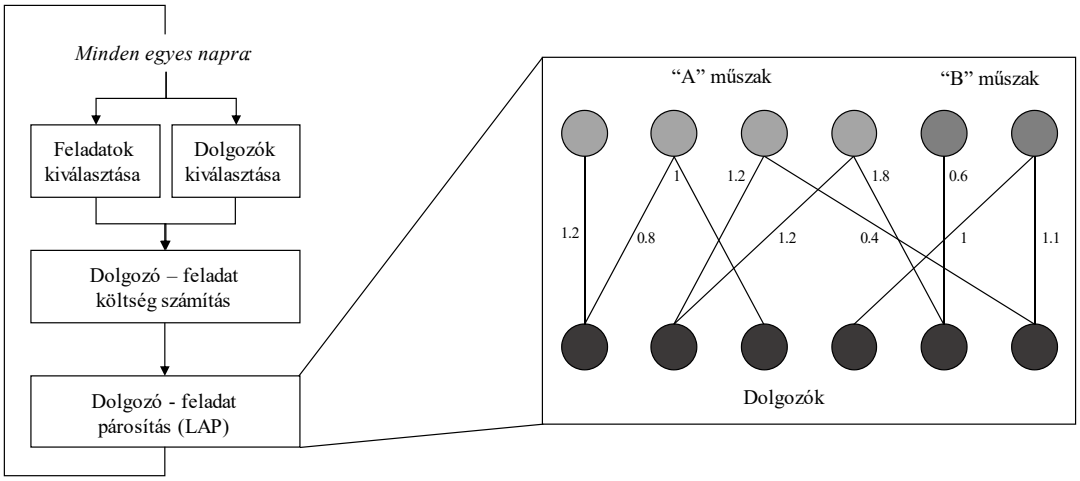
1. táblázat: Scheduling Benchmarks tesztesetek és legjobb ismert eredmények

Teszteset	Hetek	Dolgozók	Műszak típusok	Legjobb ismert pontszám	Legjobb elért pontszám automatikus beosztástervezővel
1	2	8	1	607	**607**
2	2	14	2	828	**828**
3	2	20	3	1001	**1001**
4	4	10	2	1716	**1716**
5	4	16	2	1143	**1143**
6	4	18	3	1950	**1950**
7	4	20	3	1056	**1056**
8	4	30	4	1300	**1300**
9	4	36	4	439	**439**
10	4	40	5	4631	**4631**
11	4	50	6	3443	**3443**
12	4	60	10	4040	**4040**
13	4	120	18	1348	**1348**
14	6	32	4	1278	**1278**
15	6	45	6	3823	3834
16	8	20	3	3225	**3225**
17	8	32	4	5746	**5746**
18	12	22	3	4459	**4459**
19	12	40	5	3148	3149
20	26	50	6	4743	4943
21	26	100	8	20868	21159
22	52	50	10	24064	33155
23	52	100	16	2765	17428
24	52	150	32	?	48777

3 Progresszív algoritmus

Munkánk során megterveztünk egy algoritmust a beosztástervezés problémájára mely progresszív módon, a tervezési időszakon előre haladva, napról napra készíti el a végleges beosztást. Épp ezért időben előre haladó algoritmusnak is szoktuk hívni. A módszer legnagyobb előnye az, hogy jelentősen csökkenti a keresési teret, azonban a gyakorlati megvalósítás során ez oda is vezethet, hogy nem találja meg az optimális beosztást. A 3.3-as alfejezet részletesebben is bemutatja a keresési tér, az optimális megoldás és az algoritmus kapcsolatát.

Nézzük végig az algoritmus főbb lépéseit egyetlen nap beosztásainak a megtervezése során, melyhez segítséget nyújt az 1. ábra. A célunk az, hogy az adott napi feladatokhoz megtaláljuk a legjobban megfelelő dolgozókat. Ehhez ki kell választani a tervezési időszakra megadott igények alapján az adott napra vonatkozó feladatokat és azokat a dolgozókat, akiket be lehet osztani aznapra. Ezután egy jósági mutatót fogunk kiszámolni minden egyes dolgozó-feladat pároshoz a Scheduling Benchmarks által megadott szabályok alapján, amik között vannak olyanok is, amik a teljes tervezési időszakra vonatkoznak. A jósági mutatót nevezzük el a dolgozó költségének az adott feladat elvégzéséhez. A költségek alapján már egyszerűen ki lehet választani a feladatokhoz leginkább megfelelő dolgozókat, az algoritmus pedig tovább léphet a következő napra.



1. ábra: a progresszív algoritmus főbb lépései

A dolgozók és feladatok egymáshoz rendelését részletesebben kibontva, felírhatjuk a két halmazt egy páros gráf két csúcshalmazaként, a költségeket pedig a

csúcsok között vezető élek súlyaként. Ebben a gráfban szeretnénk minél több élet kiválasztani, hogy minél több feladat el legyen végezve, de úgy, hogy az élek súlya minimális legyen az összes lehetséges módon kiválasztható élek halmazából, vagyis az optimális beosztásokat szeretnénk kiválasztani. Végül egy feladathoz legfeljebb egyetlen dolgozót szeretnénk hozzárendelni, és fordítva, azaz egyetlen dolgozó legfeljebb egy feladathoz lehet beosztva egy napon. A gráfban való keresés feladatát tömören összefoglalva tehát, minimális összsúlyú teljes párosítást keresünk páros gráfban. Ez pont megfelel a lineáris hozzárendelési probléma definíciójának (linear assignment problem, LAP), aminek az Egerváry-algoritmus egy jól bevált megoldása, a progresszív algoritmus is ezt használja a működése során.

3.1 Súlyfüggvények

A dolgozók és feladatok párosításához használt gráf élsúlyainak meghatározására különböző súlyfüggvényeket definiáltunk. Ezek feladata, hogy a dolgozó-feladat párokhoz a szabályok figyelembevételével egy jósági mutatót, azaz költséget rendeljenek. A fejezet alpontjai egy-egy súlyfüggvényt mutatnak be részletesebben a megvalósított algoritmusból.

3.1.1 Kompozit súlyfüggvény

Egy súlyfüggvény tipikusan egyetlen követelmény vagy nagyobb koncepció mentén számolja ki a jósági mutatót, ezért több súlyfüggvényt is használ az algoritmus. A párosításhoz használt gráf éleihez viszont csak egyetlen számot lehet hozzárendelni, ezért a súlyfüggvények értékét aggregálni kell. Ezt a feladatot látja el a kompozit súlyfüggvény.

Az aggregáláshoz mértani közép szerinti átlagolást használunk, kiegészítve azzal a feltétellel, hogy ha akármelyik súlyfüggvény w_{max} maximális súlyt ad, akkor az aggregátum is w_{max} lesz. Erre azért van szükség, mert a szabályok jelentős része olyan megkötéseket fogalmaz meg, amik valamilyen feltétel teljesülése esetén tiltják a dolgozó beosztását egy feladatra. Ezzel a módszerrel meg tudjuk akadályozni egy dolgozó beosztását még akkor is, ha a párosítás eredményhalmazában egyébként szerepel ez az él például azért, mert nincs más dolgozó, aki el tudná végezni az adott feladatot. Az algoritmusunk egy szűrést végez el a párosítás eredményén, és csak azokat a beosztásokat

adja hozzá a végleges beosztások halmazához, amelyek élsúlya a párosítás során kisebb, mint w_{max}.

A kompozit súlyfüggvény működését az alábbi egyenlet szemlélteti:

$$W(p,t,d) = \begin{cases} w_{max}, & ha\ \exists\, w_i(p,t,d) = w_{max}, i = 1..n \\ \sqrt[n]{\prod_i^n w_i(p,t,d)} & egyébként \end{cases},$$

ahol:

- p jelöli a dolgozót, t a feladatot, d pedig a napot,
- $W(p,t,d)$ az aggregátum, vagyis a $p-t$ páros élsúlya d napon,
- n jelöli a súlyfüggvények darabszámát,
- $w_i(p,t,d)$ az i. súlyfüggvény,
- w_{max} pedig a legnagyobb kiadható élsúly.

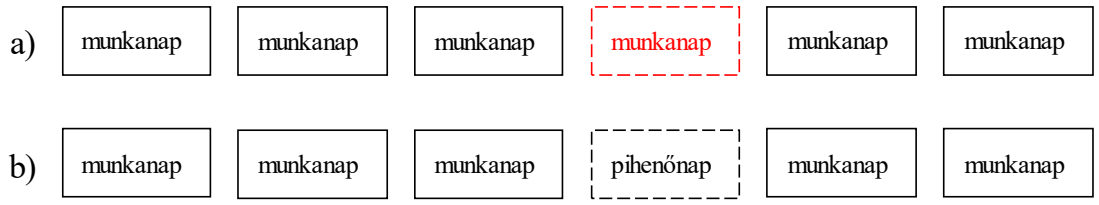
A mértani közép használata garantálja, hogy ne keletkezhessen a minimum és maximum súlyértékek intervallumán kívül eső szám, illetve alkalmas normalizált értékeknek az átlagolására [17].

3.1.2 Minimális és maximális egymást követő munkanapok száma

A címben említett szabályhoz tartozó súlyfüggvény feladata, hogy megakadályozza a dolgozó beosztását abban az esetben, ha többet, vagy kevesebbet dolgozna a munkarendjében előírt napokhoz képest. Emellett az erőforráshiányos környezetre való tekintettel törekszik arra, hogy a dolgozók beosztás sorozatai minél jobban közelítsenek a munkarendjükben meghatározott egymást követő maximális munkanapok számához. A szabály súlyfüggvénye négy főbb részre bontható.

A Scheduling Benchmarks szabályai alapján az első súlyozási szempont a maximális egymást követő munkanapok számának betartása. Például, ha legfeljebb öt napot dolgozhat egymás után a dolgozó, és a soron következő beosztás már a már a hatodik napon lenne, akkor w_{max} súlyozással megakadályozhatjuk a dolgozó beosztását. Azonban a szabály betartása során nem csak a tervezési időszakból elmúlt napokat kell figyelembe venni, hanem a jövőbeli, még hátralévő napok beosztásait is – mint ahogy a 2. ábra: a tervezési időszakból hátralévő beosztások figyelembevétele szempontjából egy

jó és rossz tervezés mutatja –, mert az algoritmus több körös tervezésre is fel van készítve, ezért nem mindig indul üres tervezési időszakkal. Tehát előfordulhat, hogy még csak három munkanapja volt az aktuális beosztás-sorozatban a dolgozónak, de a következő két nap van még egy-egy beosztása. Az ábra a) esete szerint, ha megkapná a kérdéses, szaggatott vonallal jelölt napra a beosztást, akkor már 6 napot dolgozna egyhuzamban. Erre a napra muszáj pihenőnapot adni a b) esetnek megfelelően. A több körös tervezést részletesebben a 3.5 fejezetben mutatjuk be.



2. ábra: a tervezési időszakból hátralévő beosztások figyelembevétele szempontjából egy jó és rossz tervezés

A súlyfüggvény második fő súlyozási szempontja a minimális egymást követő munkanapok számának betartása., Vegyük például, hogy legalább két napot kell dolgozni egymás után. Itt már nem olyan egyszerű a súlyozás, nem elegendő egy w_{min} súlyt adni a dolgozónak, ami garantálná, hogy megkapja a beosztást. A probléma abban gyökeredzik, hogy valójában lehetetlen egy beosztás meglétét garantálni, hiszen két dolgozó közül melyiket ossza be az algoritmus egyetlen feladatra, ha valamilyen szabály betartása érdekében mindkettőt köteles lenne beosztania? Ezért csak egy nagyon alacsony, az alapértelmezett súlykonstans töredékével, $\frac{w_{def}}{100}$ értékkel súlyoz a függvény, amikor már legalább egy napja dolgozik a beosztott, de még nem érte el a minimális egymást követő munkanapjainak számát.

A harmadik súlyozási szempont, amikor már elérte a minimális napok számát, de még nem lépte túl a maximálist. Ilyenkor $\frac{w_{def}}{2}$ értékkel igyekszik a függvény preferálni a dolgozót azokkal szemben, akik még nem kezdtek el dolgozni. Ezzel törekszik arra az algoritmus, hogy maximalizálja az erőforrások takarékos felhasználását az alapvetően erőforráshiányos környezetben.

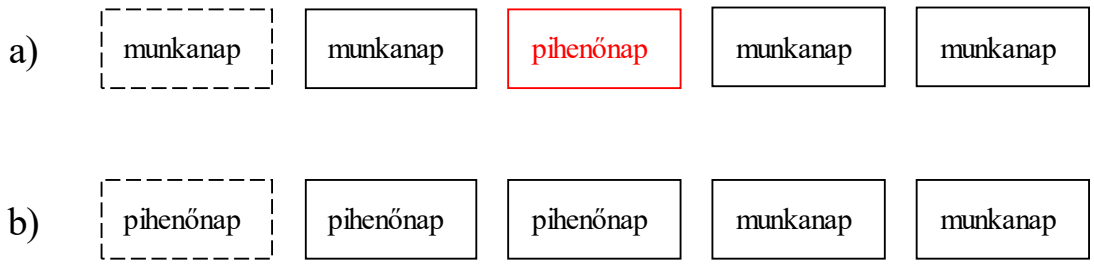
Végül a negyedik szempont a dolgozó első lehetséges munkanapjának súlyozása egy beosztás-sorozatban, szintén az erőforrás-kihasználtság maximalizálása érdekében. Az elkövetkezendő pár napra való előre tekintéssel könnyen javítható az ideális dolgozó

kiválasztása. A súlyfüggvénnyel megvizsgáljuk, hogy az elkövetkező maximálisan dolgozható egymást követő munkanapokból hány nap nem érhető el biztosan a dolgozó, illetve, hogy beleesik-e olyan hétvégi nap, amelyiken a hétvége szabály miatt nem dolgozhat. Az így meghatározott lehetséges munkanapok és maximális munkanapok aránya alapján súlyozzuk a dolgozót, tehát jobban preferálunk egy olyan dolgozót, aki az 5 napjából mind az 5-ön dolgozhat azzal szemben, aki csak 4-et. Hasonló feltételvizsgálatokkal kiszűrhetjük azokat a beosztás sorozatokat is, ahol a dolgozó garantáltan nem tudná elérni a minimális egymást követő munkanapjainak számát.

3.1.3 Minimális egymást követő pihenőnapok száma

A szabály súlyfüggvénye hasonló feladatot kell, hogy ellásson az előző, 3.1.2-es fejezet súlyfüggvényének maximális egymást követő munkanapok számának esetéhez, mert mindkét esetben bizonyos beosztások elkerülése a cél.

Másképp kell figyelni viszont a tervezési időszak hátralévő részében szereplő beosztásokra, mert előfordulhat, hogy a mai napnak kötelezően pihenőnapnak kell lennie, mert a következő beosztás sorozat elé már nem fér be minimális munkanapnyi beosztás és utána még a minimális pihenőnapok, mint ahogy a 3. ábra is látszik.



3. ábra: egy jó és rossz példa a pihenőnapok és munkanapok sorozatára

Az ábra mindkét esetében a szaggatott keretes napra kell meghatározni, hogy pihenőnap, vagy munkanap legyen, és a dolgozónak legalább 2, legfeljebb 3 napot szabad egymás után dolgoznia, 2-t pedig legalább pihennie. Az a) esetben, mivel mára beosztást kapott a dolgozó, ezért a következő napja is kötelezően munkanap lesz, viszont így csak egy napot tud pihenni a következő, 4. napon kezdődő beosztás sorozata előtt, ezért ez egy helytelen beosztás. A b) eset lesz a megfelelő beosztás ebben az esetben.

A mai napnak viszont nem feltétlen kell pihenőnapnak lennie, ha a dolgozó legfeljebb 5 napot dolgozhat, mert a 3. napra is beosztva, pont kijön a maximális 5

egymást követő munkanapja. Ugyanakkor megint előjön az a probléma, hogy nem garantálható, hogy a dolgozó megkap egy beosztást, tehát alakulhat még úgy a dolgozó 3. napja, hogy pihenni fog, és végül helytelen lesz a beosztás.

3.1.4 Maximális műszakok száma

A szabályhoz tartozó súlyfüggvény feladata, hogy a dolgozó ne kaphasson több beosztást egy adott műszak típusból, mint amennyit a munkarendje meghatároz. Ehhez számon kell tartanunk, hogy eddig hány beosztása volt a dolgozónak az egyes műszakokból a teljes tervezési időszak alatt. Ha az éppen kérdéses elvégzendő feladattal együtt már meghaladnánk a határértéket, akkor w_{max} súlyt adunk a dolgozónak, egyébként w_{def}-et. A súlyfüggvény képlete:

$$w(p,t,d) = \begin{cases} w_{max}, & ha\ workedShiftCount_{p,t} + 1 > maxShift_{p,t} \\ w_{def} & egyébként \end{cases},$$

ahol:

- $workedShiftCount_{p,t}$ jelöli, hogy p dolgozónak hány beosztása van t feladathoz tartozó műszaktípusból a teljes tervezési időszak alatt,
- $maxShift_{p,t}$ jelöli a p dolgozó munkarendjében meghatározott maximális műszakszámot t feladat műszaktípusához.

3.1.5 Minimális pihenőidő

A súlyfüggvényhez tartozó Scheduling Benchmarks szabály alapján a dolgozó két egymást követő beosztása között el kell, hogy teljen legalább annyi idő, mint a munkarendjében meghatározott minimális pihenőidő – tipikusan 11 óra.

A szabály betartásához ki kell számolnunk, hogy az előző beosztás vége és az éppen kérdéses feladat kezdete között mennyi idő telt el, és le kell ellenőrizni a kérdéses feladat vége és a következő beosztás kezdete között eltelt időt is. A súlyfüggvény képlete:

$$w(p,t,d) = \begin{cases} w_{max}, & ha\ assignmentEnd_{p,d-1} - shiftStart_t < restPeriod_p \\ w_{max}, & ha\ shiftEnd_t - assignmentStart_{p,d+1} < restPeriod_p \\ w_{def} & egyébként \end{cases},$$

ahol:

- $assignmentStart_{p,d}$, illetve $assignmentEnd_{p,d}$ rendre p dolgozó d napi beosztásának kezdete és vége,

- $shiftStart_t$ és $shiftEnd_t$ a t feladathoz tartozó kezdő- és végidőpont,
- $restPeriod_p$ a p dolgozó munkarendjében meghatározott minimális pihenőidő hossza.

3.1.6 Műszak preferenciák

A Scheduling Benchmarks műszak preferenciák szabálya alapján súlyoz a soron következő függvény. A dolgozók preferenciái meghatároznak egy súlyértéket is, ami a végleges beosztás jóságának kiértékelése során akkor jár, ha nem teljesült az adott preferencia. Ezért a súlyozás során ezt az értéket is figyelembe vesszük, de arányosítani kell az algoritmusban használt súlyozási rendszerhez, nehogy túlzottan megnövelje a dolgozók súlyát a többi szabályéhoz képest. A súlyfüggvény képlete:

$$w(p,t,d) = \begin{cases} w_{def} + s\left(shiftOffRequest_{p,t,d}\right), & ha\ \exists shiftOffRequest_{p,t,d} \\ w_{def} - s\left(shiftOnRequest_{p,t,d}\right), & ha\ \exists shiftOnRequest_{p,t,d} \\ w_{def} & egyébként \end{cases},$$

ahol:

- $shiftOffRequest_{p,t,d}$ és $shiftOnRequest_{p,t,d}$ p dolgozó preferenciája d napon t feladat műszaktípusára,
- $s(x)$ a konvertáló függvény Scheduling Benchmarks műszak preferencia súlyból progresszív algoritmuséba.

3.1.7 Minimális és maximális összmunkaidő

A dolgozó munkarendje meghatározza, hogy a teljes tervezési időszak alatt legalább és legfeljebb hány percet dolgozhat. A korábbi súlyfüggvényekhez hasonlóan a maximum érték betartása ennél a súlyfüggvénynél is egyszerű az eddig dolgozott percek nyilvántartásával és maximális súlyozással, azonban a minimális teljes munkaidő betartása nehézségekbe ütközik.

Kezdetben úgy oldottuk meg a feladatot, hogy alacsonyabb súlyt kaptak azok a dolgozók, akik még nem érték el a minimális munkaidejüket, viszont ez több szempontból sem szerencsés. A tervezés kezdetén mindenki ugyanannyival olcsóbb, ezzel nyilván nem javítunk az algoritmuson. Csupán akkor tudja kifejteni a hatását egy adott dolgozónál, amikor a többi lehetséges jelölt már teljesítette a minimum összmunkaidejét. Továbbá belefutottunk olyan esetekbe, hogy egy dolgozó nem kapta meg a tervezési időszak elején

az optimális beosztásait, amitől aztán a hátralévő időszakban csak nagyon rövideket tudott dolgozni, mert állandóan valamilyen másik szabályba ütközött, és végül nem lett meg a minimális összmunkaideje. Tehát előfordulhat olyan, hogy egy dolgozó első napi beosztása egyértelműen meghatározza a tervezési időszakra eső teljes beosztását, ami ebben a konkrét esetben épp nehézségeket okozott, azonban a detektálása rengeteget is tud segíteni az optimális végleges beosztás elkészítésében.

Az iménti problémát ideiglenesen ki tudtuk úgy küszöbölni, hogy minden egyes lehetséges beosztás előtt letervezzük a dolgozó hátralévő időszakát ezzel az adott napi lehetséges beosztással együtt, hogy kiderüljön, eléri-e a minimális összmunkaidejét, azonban könnyen látható, hogy ez egy rendkívül rosszul skálázódó megoldás. Célszerű lenne az ötletet tovább fejleszteni, hogy a teljes tervezés megkezdése előtt meghatározzunk egy ideális beosztás sorozatot a dolgozóknak, ezáltal csak egyszer kéne végrehajtani ahelyett, hogy minden egyes lehetséges beosztásra lefut újra és újra. Azonban amint eltérünk az előre kiszámolt ideális beosztástól, lehet újra számolni a hátralévő időszakra, illetve változó hosszúságúak lehetnek a műszakok, kérdés, hogy melyikkel számoljunk az ideális beosztás során? Inkább biztonságosan, a legrövidebbel? Vagy vegyük figyelembe az adott napi igényeket? Ebben az esetben már nagyon hasonlítana a tényleges beosztást tervező algoritmus logikájára, már-már duplikáltuk az eljárást. A [12] cikkben voltak hasonló irányba kísérletek, azonban nem bizonyultak célravezetőnek, illetve a 3.5-ös fejezet ismertet még alternatív megoldásokat a problémára.

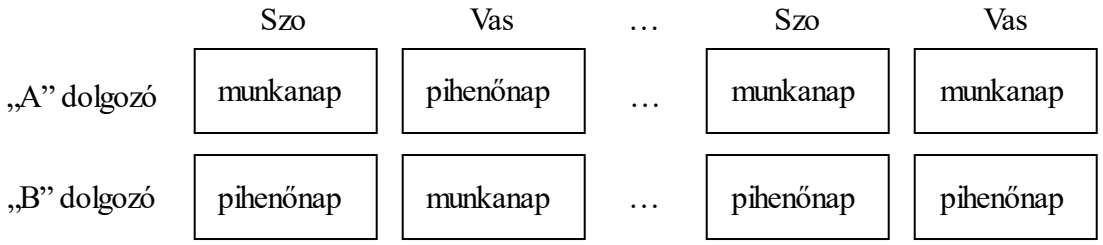
A minimális összmunkaidő kikényszerítése tehát egyelőre egy nyitott kérdés, nincs rá jól skálázódó megoldás a progresszív algoritmusban.

3.1.8 Munkával töltött hétvégék maximális száma

Az adathalmazhoz tartozó szabály alapján egy dolgozó legfeljebb a munkarendje szerint meghatározott számú hétvégén dolgozhat, ahol hétvégi munkavégzésnek minősül, ha szombaton, vagy vasárnap, vagy mindkét nap van beosztása. A most következő súlyfüggvénnyel ez a szabály alapján súlyozunk.

A súlyfüggvénnyel garantáljuk, hogy a dolgozó nem dolgozik több hétvégét, mint a munkarendjében meghatározott hétvégék száma. Ezen felül igyekszünk optimálisabb beosztást elérni úgy, hogy megdrágítjuk azokat a dolgozókat, akik nem tudnak mindkét hétvégi napon dolgozni. Erre azért van szükség, mert a hétvégi munkanapok

szempontjából is egy erőforráshiányos környezettel állunk szemben. A kezdeti tapasztalataink alapján a tervezési időszak vége felé közeledve, egyáltalán nem kaptak beosztásokat a dolgozók a hétvégi napokra. Ez könnyen előfordulhat a 4. ábra szemléltetett eset miatt.



4. ábra: egy-egy példa az erőforrás pazarló és erőforrás takarékos hétvégi beosztásra

Az első hétvégén mindketten dolgoznak, de felváltva: egyik nap az egyik dolgozik, másik nap a másik. Emiatt mindkét dolgozónak elhasználtuk egy hétvégéjét. A második hétvége mutatja az erőforrás takarékos beosztást, ahol csak az egyik dolgozó hétvégéjét használtuk fel. A súlyfüggvény képlete:

$$w(p,t,d) = \begin{cases} w_{max}, & ha\ (Sat(d) \vee Sun(d)) \wedge wkndCount_p \geq maxWknd_p \\ w_{singleWkndWork}, & ha\ Sat(d) \wedge CanWorkTomorrow(p) \\ w_{singleWkndWork}, & ha\ Sun(d) \wedge \exists assignment_{p,d-1} \\ w_{def} & egyébként \end{cases},$$

ahol:

- $Sat(d)$ és $Sun(d)$ függvények igazzal térnek vissza, ha rendre szombati, illetve vasárnapi nap d,
- $wkndCount_p$ jelöli p dolgozó munkával töltött hétvégéinek számát,
- $maxWknd_p$ jelöli p dolgozó munkarendjében meghatározott maximális munkával tölthető hétvégék számát,
- $CanWorkTomorrow(p)$ függvény igazzal tér vissza, ha p dolgozónak lehet beosztása a következő nap,
- $w_{singleWkndWork}$ jelöli az egyetlen munkanappal töltött hétvégéért járó súlyt.

3.2 Állapotgörgetés

A súlyfüggvények bemutatása során felmerültek különböző dolgozói állapotot leíró változók, amiket veszteséges újra és újra kiszámolni. Ehelyett érdemes őket görgetni, ráadásul ez szépen illeszkedik is az algoritmus időben előre haladó tulajdonságához.

Az állapotváltozók túlnyomó részét előrefelé görgetjük, azonban vannak olyanok is, amiket előre kiszámolunk és ahogy halad előre a tervezés, úgy csökkentjük az értékét.

3.3 Keresési tér, optimális megoldás

Ebben a fejezetben összehasonlítjuk a progresszív algoritmus keresési terének méretét egy képzeletbeli naiv beosztástervezőével, és rámutatok azokra a lényeges pontokra, amitől egy igazán erős alternatívát jelent a gyakorlatban. Végül kitérünk a megoldás hátrányaira is, és hogy miképp lehet ezeken javítani.

Először is vezessük be a naiv beosztástervező algoritmusunkat: vegyük az összes lehetséges feladat-dolgozó párost, azaz az összes lehetséges beosztást, majd ezeket kombináljuk az összes lehetséges módon, amivel előállítottuk az összes lehetséges végleges beosztást. Már csak annyi van hátra, hogy mindegyik végleges beosztáshoz hozzárendelünk egy jósági mutatót, és megkeressük a minimálishoz tartozó végleges beosztást.

A dolgozók és feladatok összes párosítása, azaz az összes lehetséges beosztás:

$$|A| = t_{count} \cdot p_{count},$$

ahol:

- A jelöli a beosztások halmazát,
- p_{count} a dolgozók száma,
- t_{count} a tervezési időszakba eső összes feladat száma.

A beosztások összes lehetséges részhalmazának – azaz az összes végleges beosztás – számossága:

$$|S| = 2^{|A|},$$

ahol:

- S jelöli a végleges beosztások halmazát.

A progresszív algoritmus két lényeges ponton tér el a naiv megközelítésétől: az egyik, hogy kisebb – napnyi – egységekben keresi az optimális beosztásokat:

$$\sum_{i=1}^{n} 2^{t_i \cdot p_{count}} = \sum_{i=1}^{n} |S_i| \leq |S| = 2^{t_{count} \cdot p_{count}},$$

ahol:

- n a tervezési időszak napjainak száma,
- S_i az i-ik nap beosztásainak összes lehetséges részhalmaza,
- t_i az i-ik nap feladatainak száma,
- t_{count} pedig felírható úgy is, hogy $t_{count} = \sum_{i=1}^{n} t_i$.

A másik lényeges eltérés a beosztások kiválasztásában – a szelekciós műveletben – van, amit a napokra való felbontás, illetve a Scheduling Benchmarks egyik szabálya tesz lehetővé. A szabály kiköti, hogy egy dolgozónak legfeljebb egyetlen beosztása lehet egy napon. Tehát az összes napi beosztás legenerálása és kombinálása helyett bevethetjük a lineáris hozzárendelés problémáját (LAP), amire az Egerváry algoritmus futásideje $O(n^4)$, ahol n jelöli a gráf csúcsainak számát. A futásidő a következőképp alakul:

$$\sum_{i=1}^{n} (t_i + p_{count})^4.$$

A LAP bevetésével megváltozik a jósági mutató számítása is, hiszen értelmét veszti a teljes beosztásra kiszámolni, amíg nem készültek el a végleges beosztások az összes napra. Ehelyett az Egerváry algoritmusnak szüksége van élsúlyokra, amik nyilván nem egyeznek meg a teljes beosztás jósági mutatójával. A progresszív megközelítés hátránya is ezen a területen bukkan elő: valójában egy lokális optimalizáló algoritmusról van szó, a globális optimum megtalálását nem garantálja. Lehet, hogy egy adott napra megtalálja az ideális beosztástokat, de hiányoznak a tervezési időszak hátralévő részéből, a még meg nem tervezett napok beosztásából kinyerhető információk, amik alapján globálisan is optimális végleges beosztást tudna készíteni. Gondolhatunk erre úgy is, hogy egy globális optimum szempontjából helytelen beosztást is beválaszthat a véglegesek közé, amivel kizárta a keresési tér azon ágát, amin az optimális található. A probléma elméleti síkon kiküszöbölhető akár precízebb súlyfüggvényekkel is, azonban ez már nem képezi a dolgozat tárgyát.

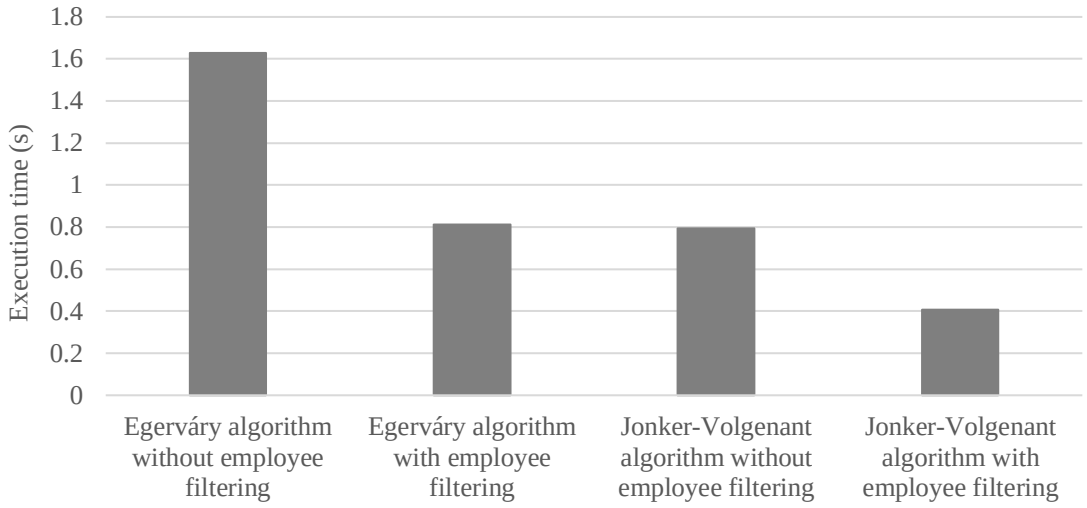
A progresszív algoritmusnak ennek ellenére is van jelentősége, mert a naív megoldásokhoz – és mint később meglátjuk, más megoldásokhoz képest is – óriási előnnyel rendelkezik a futásidő tekintetében, ami összetettebb beosztástervezési feladatoknál akár jelentheti azt is, hogy immár értelmezhető időn belül eredményt kapunk a problémára, még ha jóság szempontjából csak közelítő is.

3.4 Dolgozók előzetes szűrése

A progresszív algoritmust futásidejét jelentős mértékben tovább csökkenti a dolgozók kiválasztási logikájának finomhangolása, mert így kisebb méretű gráfon kell dolgoznia az Egerváry algoritmusnak, aminek a futásideje a csúcsok számának függvényében $O(n^4)$.

A kiválasztási logikában eleinte csak az játszott szerepet, hogy melyik dolgozó érhető el az adott napon, de később a súlyfüggvényekből is átkerültek olyan kódrészletek, amik maximálist súllyal térnek vissza. Azonban nem lehet az összes maximális súlyozású logikát kiemelni, csak azokat, amik nem függenek a műszak típusától. Ilyen például a maximum összmunkaidő figyelése hiszen, ha egy dolgozó már nem dolgozhat többet a teljes tervezési időszak hátralévő részében, akkor felesleges megpróbálkozni a beosztásával. Viszont nem emelhető ki az a logika, amelyik a nem beosztható műszakokra ad maximális súlyt a dolgozónak, mert egy nap vegyesen előfordulhatnak a dolgozó által elvégezhető és nem elvégezhető műszakok is. Arra ugyan rá lehetne ellenőrizni, hogy van-e egyáltalán olyan műszakra igény az adott napon, amelyiket el tudja végezni a dolgozó, de továbbra is ugyanúgy le kéne ellenőrizni súlyfüggvénnyel, hogy a vegyes dolgozókat továbbra is kiszűrjem, bár a dupla ellenőrzés egy nagyobb bemenetre minden bizonnyal még mindig alacsonyabb futásidőt eredményezne, mint az Egerváry $O(n^4)$-es futásideje. Azonban az algoritmus jelenlegi állapotában ez mikrooptimalizációnak számít, így nem valósítottam meg.

A dolgozók szűrésének az algoritmus futásidejére gyakorolt hatását [11] részletesebben is tárgyalja, amit jól szemléltet az 5. ábra segítségével.



5. ábra: dolgozók szűrésének hatása a beosztástervezés futási idejére Egerváry, illetve Jonker-Volgenant algoritmusának használata esetében [11]

3.5 Többkörös tervezés

A 3.3-as fejezetben említett hátrányból le lehet faragni úgy is, ha többször, különféle aspektusok mentén tervezünk az időszakra. Ehhez azonban fel kell készíteni a súlyfüggvényeket és az állapotgörgetést arra, hogy a tervezési időszak hátralévő részében is előfordulhat beosztás, ezeket a kapcsolódó fejezetek részletezik.

A leginkább szembetűnő probléma az volt, hogy sok dolgozó nem tudta elérni a minimális összmunkaidejét annak ellenére, hogy a tervezési időszak vége felé drasztikusan megnőtt a kielégítetlen műszak igények száma és látszottak is az eredményeken, hogy a hétvégék nincsenek gazdaságosan felhasználva, túlnyomó részüknél csak egyik nap volt beosztása a dolgozónak. Emiatt be kellet hívni a másik hétvégi napra egy másik dolgozót, amitől mindkettő elhasznált egy-egy hétvégét, pedig elég lett volna csak az egyikőjük hétvégéjét elhasználni. Ezért a beosztástervezést azzal kezdtem, hogy kizárólag a hétvégi napokra tervezek, annak reményében, hogy más kényszerek nem fogják korlátozni a dolgozókat abban, hogy mindkét hétvégi napot megkapják, ha már be lettek osztva az egyikre. Miután elkészült a hétvégi beosztás, újra ráterveztem a teljes időszakra, az eddig megszokott módon. Meglepetésemre egyáltalán nem javított a végleges beosztáson, sőt, jelentősen rontott, mert immár a többi kényszer ütközött különféle korlátokba, például az egybefüggő munkanapok száma és

pihenőnapok száma volt, hogy nem fért be hétközben, ha még egy pihenőnapja is volt a hét során.

Megoldást végül az jelentett a minimális összmunkaidő problémájára, hogy a normál tervezés után még addig osztottam be a dolgozókat, amíg el nem érték a minimális összmunkaidejüket akár annak árán is, hogy több beosztást készítek egy napra, mint az eredeti igény. Ez nem akkora probléma, mert nagyon kevés büntetőpont jár érte. Továbbá egy újabb körben a dolgozók egyetlen beosztásos hétvégéit összevontam dupla beosztásossá ott, ahol kemény követelményt nem szegtem ezzel, majd a teljesen felszabadult hétvégék között oszlattam el az igényszám felett tervezett beosztásokat. Ezzel sikerült az első 8 tesztesetnél megszüntetni a minimális összmunkaidő megszegését, ugyanakkor az eredmény nagyon jól mutatja azt is, hogy messze nem tökéletes megoldásról van szó, inkább csak kisebb javításokat lehet vele eszközölni.

4 IP Solver

Az Scheduling Benchmarks feladatának felírás IP feladatként három lépésre bontható:

- a modell felépítése
- a kényszerek felírása a modellre
- a célfüggvény megfelelő definiálása

A modellünk egy teljes páros gráf, amelyben a pontosztályokat a dolgozók és a műszakok alkotják. Az élek súlyozását feleltettük meg annak a beosztásnak, amelyet a Scheduling Benchmarks feladat egy lehetséges megoldásának tekintünk. A pontos modellt a Scheduling Benchmarks által megfogalmazott feladatleírás alapján további feltételek segítségével alkottuk meg..

A kényszerek és a célfüggvény felépítése helyenként összefüggenek. A Scheduling Benchmarks által felállított kényszerek rendszere és büntetőpontok rendszere közösen határozzák meg, hogy egy adott megkötés befolyásolja-e a célfüggvényt is, vagy sem.

Az IP feladat egyenlőtlenségei esetén csak azokat a változókat említjük, amelyek közvetlenül jelen vannak egy adott egyenlőtlenségben. Továbbá a Scheduling Benchmarksban előforduló megnevezéseket magyar nyelven használjuk. Egy új kifejezés bevezetésekor a zárójelben dőlt betűvel feltűntetjük a Scheduling Benchmarks által használt kifejezés.

4.1 A modell felépítése

A modell alapja egy teljes páros gráf illeszkedési mátrixa. A Scheduling Benchmarks magas szinten igényekkel (*Cover Requirements*, fedési igény) dolgozik, ami meghatároz egy napra (*Day*), egy műszaktípusra (*Shift Type*) vonatkozó igényeket. A műszaktípus pedig meghatározza a műszak időszakát. A műszak kifejezést innentől kezdve arra a komplex objektumra értjük, ami meghatározza, hogy melyik napon, milyen típusú műszakra (ez által, mikor és mennyi időre) van szükség dolgozó(k)ra. Fontos megjegyezni, hogy az igényekhez egy preferált minimum és egy preferált maximum dolgozói szám is hozzátartozik.

Ez alapján a hozzárendelési változóink száma innentől AVC (Assignment Variable Count)

$$AVC = EC * SC,$$

ahol EC (Employee Count) a bemenetben szereplő dolgozók számát, SC (Shift Count) a bementeben szereplő műszakok számát jelöljük.

A gráf éleit reprezentáló változókat jelölje e_{ijk}, $O \leq i < EC$, $O \leq j < DC$, $O \leq k < SC$ ahol SC a tervezési időszakba eső napok száma, i a dolgozó indexe, j a tervezési időszakra eső napok indexe, k pedig a műszak indexe. Ennek a reprezentációnak az alapjául egy az alábbi cikk szolgált [18].

A gráf éleinek súlyozása, azaz az IP feladat e_{ijk} változóinak értékkészletére két feltétel áll fenn:

$$\forall i \forall j \forall k (0 \leq e_{ijk} \leq 1, e_{ijk} \in \mathbb{Z})$$

Ez feltétel az IP feladat felírásában 2*AVC darab egyenlőtlenséget jelent az e_{ijk} változók értékkészletére. Az egészértékű feltételre persze azért van szükség, mert lineáris programozási (innentől LP) feladatként vizsgálva nem csupán értelmetlen eredményeket kapnánk (pl.: i. ember a j. műszakra beosztásra került 0.79 részben), de matematikailag bizonyított, hogy nem lehet következtetni az LP feladat megoldásáról az IP feladat megoldására általános esetben.

4.2 Kényszerek felírása a modellre

A Scheluling Bechmarks megkötései két nagy csoportra bonthatók, az alapján, hogy a megkötést kötelező-e:

- erős kényszerek
- gyenge kényszerek

Az erős kényszereket mindenképp tartania kell egy megoldásnak. A Scheduling Bechmarks terminológiájában ezen kényszerek be vagy be nem tartása a probléma megoldhatóságát, fiziliblitását jelentik. Az IP feladatban ezek a kényszerek konkrét egyenlőtlenségekként jelennek meg, amik a megfelelő e_{ij} változókra közvetlenül, vagy belőlük származtatott segédváltozókra kerülnek felírásra. Továbbá erős kényszerként jelennek meg azok az implicit kényszerek, amelyek felírása szükségszerű, hogy a modell szemantikája megfelelő legyen.

A gyenge kényszerek teszik a Scheduling Benchmarks feladatait érdekessé, ugyanis ezek a megkötések megszeghetők, de megszegésük bizonyos mértékű bűntetőpontot (*penalty*) von maga után. Ezek a bűntetőpontok végül összegezve adják a Scheduling Benchmarks objektív mérőszámát, ami alapján a kapott beosztások összemérhetők. Fontos megjegyezni, hogy csak a fizibilis azaz erős kényszert nem sértő beosztásoknál van értelme ennek a pontszámnak. Az egyes gyenge kényszerekhez tartozó büntetőpontszám a célfüggvényekben koefficiensként, a büntetés előfordulása, illetve darabszáma egy változóként jelenik meg.

Az erős kényszerek a modellben:

- Napi egy beosztás megkötés
- Rögzített pihenőnap megkötés
- Rögzített munkavégzés megkötés
- Egymást követő munkanapokra vonatkozó megkötés
- Összmunkaidő megkötés
- Hétvégék számára vonatkozó megkötés

A gyenge kényszerek a Scheduling Benchmarksban:

- Munkavégzési igény megkötés
- Pihenőnapigény megkötés
- Műszakon dolgozók számára vonatkozó igény

A Scheduling Benchmarks meghatároz még néhány kényszertípust, de ezek nem relevánsak, vagy elő se fordulnak a kezdeti tesztesetekben.

Az egyértelműség kedvéért nevezéktanban is megkülönböztetjük a Scheduling Bechmarks, illetve az általunk felírt IP feladat kényszereit. A Scheduling Benchmarks kényszereit megkötéseknek nevezem, míg az mi IP feladatunkban előforduló egyenlőtlenségeket kényszereknek nevezzük.

4.2.1 Napi egy beosztás megkötés

Ahhoz, hogy ez az egész feltételrendszer értelmet nyerjen a beosztástervezés kontextusában el kell érni azt, hogy egy dolgozó ne dolgozhasson egyszerre több

műszakon. Ehhez 2*DC*EC kényszerre van szükségünk. A kényszerek az alábbi módon írtuk fel:

$$0 \leq \sum_{k=0} e_{ijk}\,, \forall i \forall j(\, O \leq i < EC, O \leq j < DC)$$

$$\sum_{k=0} e_{ijk} \leq 1, \forall i \forall j(\, O \leq i < EC, O \leq j < DC)$$

Mivel az e_{ijk} változók értékkészlete korlátozott a 0 és 1 értékekre, ezért, egy dolgozó egy napra eső műszakjainak a változók összege az adott dolgozó adott napra eső műszakjainak a számát jelöli. Ennek az összegnek a korlátozása a 0 és 1 értékekre megfelel annak, hogy egy dolgozó egy nap vagy nem dolgozik, vagy ha dolgozik, akkor csak egy műszakhoz lehet hozzárendelve.

4.2.2 Rögzített pihenőnap megkötés

A rögzített pihenőnapok megkötés (*Fixed Free Days*) eredendően arra vonatkozik, hogy egy dolgozó egy adott napon nem dolgozhat. Az inputban szereplő rögzített pihenőnapok számára vonatkozó kényszereket ffd_{ij}-vel jelöljük ($O \leq i < EC, O \leq j < DC$), ahol i a dolgozó indexe, j pedig a rögzített pihenőnap napja. Az inputban szereplő rögzített pihenőnapok számára vonatkozó kényszerek számát jelölje FFDC (Fixed Free Day Count). Mivel ez egy erős kényszer, az adott napra a dolgozóhoz rendelt műszakok száma kötelezően 0-nak kell legyen. A műszakokhoz rendeltséget jelölő változók értékkészlete a {0, 1} halmaz, a kényszerek érvényesítéséhez elegendő 2*FFDC darab új kényszer bevezetése. Ezek zárt alakban:

$$0 \leq \sum_{k=0} e_{ijk}\ , \forall \mathrm{ffd}_{ij}$$

$$\sum_{k=0} e_{ijk} \leq 1\ , \forall \mathrm{ffd}_{ij}$$

4.2.3 Rögzített beosztás megkötés

A rögzített beosztás megkötés (*Fixed Assignment*) egy erős kényszer, amely meghatároz egy dolgozó-nap-műszaktípus hármast. Ezek meghatározzák, hogy a modellem egy adott változójának kötelezően 1-es értéket kell felvennie. Ehhez pontosan egy kényszerre volt szükségünk, amely felírható az alábbi módon:

$$e_{ijk} = 1$$

4.2.4 Egymást követő munkanapokra vonatkozó megkötés

Az egymást követő munkanapokra vonatkozó megkötés (*Min Sequence* és *Max Sequence*) egy erős kényszer, ami meghatározza, hogy egy dolgozó egy adott műszaktípusból legalább és legfeljebb hány egymást követő nap dolgozhat. A műszaktípus megnevezése helyén az input definiál két speciális típust, egyiket a tetszőleges műszakra, másikat a munkaszüneti napra (műszak hiányára). Jelölje SeCC (Sequence Constraint Count) az egymást követő munkanapokra vonatkozó kényszerek számát. Ez a bemenetben nem jelenik meg egyértelműen. Az egy adott munkarendhez (Contract) tartozó egymást követő munkanapokra vonatkozó kényszerek számát meg kell szorozni a munkarendhez hozzárendelt dolgozók számával, hogy megkapjuk ezen kényszerek pontos számát. Ezzel egy jó felső becslést kaptunk, ha feltételezzük, hogy minden dolgozóhoz minden munkarend hozzárendelésre kerül. Így, ha ISeCC (Input Sequence Constraint Count) jelöli a bementben előforduló egymást követő munkanapokra vonatkozó kényszerek számát, akkor felső becslésünk:

$$SeCC \leq ISeCC * EC$$

Mivel kezelésükben a minimum és a maximum egymást követő munkanapokra vonatkozó megkötések kezelés jelentősen különbözik, a továbbiakban a szekvenciák minimumára, illetve maximumára vonatkozó igényeket jelölje rendre, MaxSe_{ijt} $O \leq i < EC, O \leq j < DC$, ahol i a dolgozó indexe, j a rögzített pihenőnap napja, t pedig a műszak típusát jelöli. Mind MinSe_{ijt}, mind MaxSe_{ijt} bekorlátozható az inputban szereplő dolgozók számával:

$$1 \leq MinSe_{ijt} \leq EC$$

$$1 \leq MaxSe_{ijt} < EC$$

Értelemszerűen a 0-s értéket nem érdemes taglalni, egyik esetben sem. A felső határ esetén a maximum meghatározásánál szintén nem hordoz információt, ha ez az érték megegyezhet a bemenetben szereplő dolgozók számával.

A Scheduling Benchmarks két speciális jelölést használ két speciális műszak típusra, ezek a „$” a tetszőleges műszaktípusra, „-” a munkával nem töltött típusra (ez jelképezi a pihenőnapokat).

A minimum és a maximum korlátok, illetve a műszak típusa alapján további három esetre bomlik ennek a kényszernek a vizsgálata, ugyanakkor a Scheduling

Benchmarks nem tartalmazza ezeknek mind a hat lehetséges kombinációját. Ezek alapján a Scheduling Benchmarksban előforduló esetek az alábbiak:

- Minimum egymást követő pihenőnapok száma
- Minimum egymást követő munkanapok száma
- Maximum egymást követő munkanapok száma

A minimum egymást követő pihenőnapok számára vonatkozó feltételek ($MinSe_{ij}$-alakú megkötések) meghatározása nem triviális.

$MinSe_{ij\$}$ értéke alapján a szekvencia hosszára van egy felső korlát. Ez $MinSe_{ij\$} = 1$ esetén triviálisan teljesül. A kényszer felírásához azt használtam ki, hogy a $MinSe_{ij\$}$-1 a leghosszabb sorozata napoknak, amin a dolgozót nem lehet beosztani. A célom az volt, hogy felismerjek minden legfeljebb $MinSe_{ij\$}$-1 hosszú 1-es sorozatot a dolgozó beosztásaiban, és megakadályozzam ezeket. Ehhez szükséges volt a napoknak legalább $MinSe_{ij\$}$+1 hosszú sorozatait tekintenem. Az utolsó szükséges állítás ehhez, hogy egy nagy $MinSe_{ij\$}$ érték elsetén is elegendő külön-külön tekinteni, hogy megjelenik-e 1, 2, …. $MinSe_{ij\$}$-1 hosszú sorozat. A munkám során csupán a $MinSe_{ij\$} = 2$ esettel dolgoztam, mert ez elegendő a kezdeti tesztesetekhez. Így a 0-1-0 szekvenciát kellett felismernem.

$MinSe_{ij\$} = 2$ -hez tartozó kényszerek (a konstans értékek szemléltetésként szerepelnek a képletben):

$$\forall i \forall k \in \{0, DC-3\} \left(-2 \leq -\sum_{j=0}^{\mathrm{SC}-1} e_{ijk} + \sum_{j=0}^{\mathrm{SC}-1} e_{ijk+1} - \sum_{j=0}^{\mathrm{SC}-1} e_{ijk+2} \right)$$

$$\forall i \forall k \in \{0, DC-3\} \left(-\sum_{j=0}^{\mathrm{SC}-1} e_{ijk} + \sum_{j=0}^{\mathrm{SC}-1} e_{ijk+1} - \sum_{j=0}^{\mathrm{SC}-1} e_{ijk+2} \leq 0 \right)$$

$$-2 \leq -1 + \sum_{j=0}^{SC-1} e_{ijO} - \sum_{j=0}^{SC-1} e_{ij1}$$

$$-1 + \sum_{j=0}^{SC-1} e_{ijDC-2} - \sum_{j=0}^{SC-1} e_{ijDC-1} \leq 0$$

A 0-1-0 szekvencia pontos felismerését végző képlet helyességét az alábbi táblázat bizonyítja, ahol az eredmény oszlop értékeit úgy kapom, ha a k. nap oszlop

értékeiből kivonom a k-1. nap illetve a k+1. nap oszlop értékeit. A kényszerekben szereplő konstans határokat is igazolja 2. táblázat.

2. táblázat: MinSe$_{ij\$}$ = 2 feltételhez tartozó kényszerek helyessége

k-1. nap	k. nap	k+1. nap	Eredmény
0	0	0	0
0	0	1	-1
0	**1**	**0**	**1**
0	1	1	0
1	0	0	-1
1	0	1	-2
1	1	0	0
1	1	1	-1

A MinSe$_{ij\text{-}}$ felírása hasonlóképp történik, annyi különbséggel, hogy ebben az esetben a pihenőnapok korlátozása miatt az nulla érték sorozatait kell korlátozni. A képletet szintén csak MinSe$_{ij\text{-}}$=2-re mutatom be:

$$\forall i \forall k \in \{0, DC-3\} \left(-1 \leq \sum_{j=0}^{\mathrm{SC}-1} e_{ijk} - \sum_{j=0}^{\mathrm{SC}-1} e_{ijk+1} + \sum_{j=0}^{\mathrm{SC}-1} e_{ijk+2} \right)$$

$$\forall i \forall k \in \{0, DC-3\} \left(\sum_{j=0}^{\mathrm{SC}-1} e_{ijk} - \sum_{j=0}^{\mathrm{SC}-1} e_{ijk+1} + \sum_{j=0}^{\mathrm{SC}-1} e_{ijk+2} \leq 1 \right)$$

$$-1 \leq 1 - \sum_{j=0}^{SC-1} e_{ijO} + \sum_{j=0}^{SC-1} e_{ij1}$$

$$1 - \sum_{j=0}^{SC-1} e_{ijDC-2} + \sum_{j=0}^{SC-1} e_{ijDC-1} \leq 1$$

Ehhez a felíráshoz tartozó magyarázat hasonló, a képletben az előjelek felcserélésével, és korlátok megváltoztatásával elérhető a megfelelő 1-0-1 minta felismerése. Erre a helyességbizonyítást szintén a 3. táblázattal mutatjuk be:

3. táblázat: $MinSe_{,ij-}$ = 2 feltételhez tartozó kényszerek helyessége

k-1. nap	k. nap	k+1. nap	Eredmény
0	0	0	0
0	0	1	1
0	1	0	-1
0	1	1	0
1	0	0	1
1	**0**	**1**	**2**
1	1	0	0
1	1	1	1

A maximum egymást követő munkanapok számára vonatkozó megkötés is a Scheduling Benchmarksban csak t = $ esetben fordul elő. Ennek a megkötésnek a modellezése egy a maximumnál egy nappal nagyobb csúszó ablakkal történt. Mivel feltételezhetjük, hogy egy dolgozó egy nap csak egy műszakhoz lehet hozzárendelve. Továbbá, hogy a hozzárendeléseket modellező változók hozzárendelés esetén 1, egyébként 0 értékűek. Belátható, hogy ez a maximum igénynél pontosan egy nappal hosszabb ablakban a maximum egymást követő munkanapokra vonatkozó feltétel úgy valósul meg, hogy ezen ablakon belül eső változók összege nem haladhatja meg a maximum mértékét. Ez egy $MaxSe_{ij\$}$-val jelölt kényszerhez pontosan DC- $MaxSe_{ij\$}$-1 darab kényszert igényel. Ezek zárt alakban:

$$\sum_{k=O}^{MaxSe_{ij\$}+1} \sum_{j=0}^{x-1} e_{ijk} \leq MaxSe_{ij\$}$$

$$0 \leq \sum_{k=O}^{MaxSe_{ij\$}+1} \sum_{j=0}^{x-1} e_{ijk}$$

4.2.5 Összmunkaidő megkötés

Az összmunkaidő megkötés (*Workload*) egy dolgozóhoz a munkarendje alapján hozzárendelt minimum és maximum mennyiség, amelyek meghatározzák, hogy a teljes beosztás tervezési időszakra mennyi munkaidő rendelhető az adott dolgozóhoz. Jelölje az i indexű dolgozó minimum és maximum összmunkaidejét rendre $\mathrm{MinWL_i}$, $\mathrm{MaxWL_i}$. Ezek számossága WLC (Workload Count) minden esetben $WLC = 2 * EC$. A kényszerek az IP rendszerben a műszaktípusok által meghatározott munkaidőket koefficienskét felhasználva az ember-műszak kapcsolatot modellező változókra. A kényszerek az alábbiak:

$$\forall i(O \leq i < EC, \sum_{j=0}^{DC} \sum_{k=0}^{SC} SWL_{k^*} e_{ijk} \leq MaxWL_i)$$

$$\forall i \left(O \leq i < EC,\ MinWL_i \leq \sum_{j=0}^{DC} \sum_{k=0}^{SC} SWL_{k^*} e_{ijk} \right)$$

4.2.6 Hétvégék számára vonatkozó megkötés

A hétvégék számára vonatkozó megkötés a Scheduling Benchamarks által használ adatmodell egy kellően leegyszerűsített változata. A dolgozói munkarendekben szereplő minta (*Pattern*) és ezeken belüli illeszkedési (*Match*) feltételek kifejezőereje jelentősen bővebb ugyan, de esetünkben csupán egy egyszerű feltételt fogalmaznak meg. Ez a feltétel egyetlen paraméterrel rendelkezik, ez a minta illeszkedésének, előfordulásának maximális száma a tervezési időszakban. A minták minden esetben azonosan a hétvégi munkavégzést fejezik ki oly módon, hogy egyszeri előfordulásnak számít, ha:

- a csak szombati munkavégzés
- a csak a vasárnapi munkavégzés
- a szombati és a rá következő vasárnapi munkavégzés

Ennek a reprezentációját segédváltozókkal végeztem. Minden dolgozóra a tervezési időszakba eső hetek számának megfelelő darabszámú változóra volt szükség. Ezeknek az összegére volt maximum igény megadva, éppen ezért, ha MWVC (Maximum Weekend Variable Count) jelöli az így keletkező változók számát, MWVC (Maximum Weekend Constrtaint Count) jelöli az így keletkező kényszereke számát, akkor az alábbi

állítások igazak a megkötésekre, illetve a változókra WCC (Weekend Constraint Count) darab ilyen bemeneti kritérium esetén:

$$MWVC = EC * \frac{1}{7} EC$$

$$MWCC = EC * \frac{1}{7} EC + WCC$$

Továbbá fontos kiemelni, hogy egy dolgozóhoz egynél több ilyen megkötés nem fogalmazható meg, így WCC-re is van egy felsőbecslés:

$$WCC \leq EC$$

4.2.7 Munkavégzési igény megkötés

A dolgozók által megfogalmazott munkavégzési igény (*Shift On Request*) egy gyenge kényszer, meghatároz egy dolgozó-nap-műszaktípus hármast. Ha a meghatározott dolgozó a meghatározott napon a meghatározott műszaktípusra nem kap beosztást, akkor a súly (*weight*) által meghatározott konstans mennyiségű büntetőpont jár. Ehhez egy segédváltozót használtam, amely pontosan akkor vett fel egy 1-es értéket, amikor ez a feltétel nem teljesül, azaz amikor a dolgozó nem kerül beosztásra a megfelelő napon a megfelelő műszaktípusra. Jelölje sor_{ijk} ezt a segédváltozót. Nyilván valóan az értékkészlete megegyezik az összerendelésekre használt változó értékkészletével:

$$0 \leq sor_{ijk} \leq 1$$

A felírt kényszerek alakja:

$$e_{ijk} + sor_{ijk} = 1$$

Minden ilyen megkötéshez szükséges egy segédváltozó, illetve egy kényszer. Jelölje SORVC (Shift On Request Variable Count) a megkötéshez tartozó változók számát, SORCC (Shift On Request Constraint Count) a megkötéshez tartozó kényszerek számát. Ha SORC (Shift On Request Count) darab ilyen jellegű megkötést tartalmaz az input, akkor a keletkező változók és kényszerek száma is ugyanennyi.

$$SORCC = SORC$$

$$SORVC = SORC$$

4.2.8 Dolgozói pihenőnapigény megkötés

A dolgozók által igényelt pihenőnapok (*Shift Off Request*) egy gyenge kényszer. Meghatároz egy napot és egy műszaktípust, amely napon, ha a meghatározott műszaktípusra kerül beosztásra a dolgozó, akkor egy meghatározott súly (*weight*) szerint egyszeri konstans büntetőpont jár. Ennek a megvalósítása lehetne hasonló dolgozói munkavégzési igény felírásához, ugyanakkor észrevehető, hogy itt nincs szükség segédváltozóra. A súly által meghatározott büntetőpont pontosan akkor jár, ha van olyan $e_{ijk,}$ amelynek az értéke 1 és i, illetve k rendre a megkötésben meghatározott dolgozó és nap.

A büntetőpontok kezelését a célfüggvény definiálja.

4.2.9 Műszakon dolgozók számára vonatkozó igény

Munkavégzési igényre vonatkozó kényszerek gyenge kényszerek. Ezek úgynevezett fedési igényekként (*Cover Requirement*) jelennek meg a Scheduling Benchmarks modelljében. Egy fedési igény meghatározza, hogy adott napra, adott műszaktípushoz minimum, illetve maximum hány dolgozóra van igény. Ezen felül definiál két súly (weight) értéket, amelyek a minimum, illetve a maximum igények megsértése esetén jelennek meg büntetőpontként. A büntetőpont viszont nem egy konstans értékként jelenik meg, hanem egy ilyen jellegű megkötés megszegésénél a megfelelő felső vagy alsó korláttól való eltérés (minimum alatti, vagy maximum feletti dolgozók száma) meghatároz egy különbséget. Ezen különbség kerül megszorzásra a megfelelő súlyértékkel.

Ennek a megkötésnek a tartásához szükséges miden igényhez meghatározni két további változót, amelyek a minimum alatti, illetve a maximum feletti dolgozók számát fogják megadni. $UMin_{jk}$ jelölje a k naphoz, j műszaktípushoz tartozó igény minimuma alatti beosztott dolgozók számát, $OMax_{jk}$ pedig a k naphoz, j műszaktípushoz tartozó igény maximuma feletti beosztott dolgozók számát. Továbbá Min_{jk} és Max_{jk} jelölje a munkavégzési igényhez tartozó minimum, illetve maximum dolgozószámokat k napon, j műszaktípushoz.

Fontos kiemelni, hogy a modellemben így egy igényt adott nap (k rögzített), adott műszaktípus esetén (j rögzített) $EC + 2$ darab változó reprezentál:

$$e_{ijk} \;\; 0 \leq i < EC, UMin_{jk}, OMax_{jk}$$

A határoktól történő eltéréseket jelölő változók értékkészletét így állapítottam meg:

$$0 \leq UMin_{jk} \leq Min_{jk}$$

$$0 \leq OMax_{jk} \leq EC - Max_{jk}$$

Az alábbi kényszereket írtam fel egy adott igényhez:

$$Min_{jk} \leq UMin_{jk} + \sum_{i=0}^{EC-1} e_{ijk}$$

$$UMin_{jk} + \sum_{i=0}^{EC-1} e_{ijk} \leq EC$$

$$0 \leq \left(\sum_{i=0}^{EC-1} e_{ijk} \right) - OMax_{jk}$$

$$\left(\sum_{i=0}^{EC-1} e_{ijk} \right) - OMax_{jk} \leq Max_{jk}$$

Látható, hogy a kényszerek önmagukban nem biztosítják az $OMax_{jk}$, $UMin_{jk}$ értékekeire, hogy pontosan a Min_{jk} és Max_{jk} értékektől való eltérési értékeket vegyék fel. Amennyiben az igényre beosztott dolgozók száma az igényben megfogalmazott határokon belül esik $OMax_{jk}$, illetve $UMin_{jk}$ értékeinek 0-nak kell lennie, egyébként a határon kívül eső dolgozók száma. De ezek a kényszerek felülről nem korlátozzák ezekhez a célértékekhez a segédváltozókat, csupán alulról korlátozzák az értékeiket. Ennek a feltételnek a biztosítását a célfüggvény biztosítja.

Látható, hogy minden igényhez pontosan négy kényszerre van szükségem. Ha a bementben előforduló igények száma CRC (Cover Requirement Count), akkor a szükséges kényszerek száma CRCC (Cover Requirement Constraint Count):

$$CRCC = 4 * CRC$$

4.3 Célfüggvény definiálása

A célfüggvény egy lineáris függvénye az IP feladat változóinak. A változók jelentős része nem szerepel a célfüggvény definiálásában, ezek precízen 0 koefficienssel rendelkeznek.

A célfüggvény képzésében azok a változók jelennek meg, amelyek valamilyen gyenge kényszer megszegéséért valamennyi büntetőpontot adnak. A célom az volt, hogy itt a célfüggvény értéke pontosan a Scheduling Benchmarks pontszáma legyen. Ehhez a három különböző gyenge kényszer esetén külön-külön teljesítettem, hogy a megfelelő büntetőpontot adják.

4.3.1 Munkavégzési igények megszegése

Amennyiben egy munkavégzési igény megszegése történik, akkor az adott munkavégzési igényhez tartozó segédváltozó értéke konstans 1 lesz. Ily módon, hogy a megfelelő koefficiens ehhez a változóhoz a munkavégzési igény megkötésében szereplő súly paraméter.

4.3.2 Pihenőnapigény megszegése

Pihenőnap igény esetén nem használtam segédváltozót, hanem a beosztás tényét reprezentáló változókat használtam. Ezt megtehettem, hiszen nincs másik célfüggvény, ami direkt ezt a változót használná. A koefficiense miden dolgozói pihenőnapigényhez a megkötésben szereplő súlyérték.

4.3.3 Műszakon dolgozók számára vonatkozó igény megszegése

A műszakon dolgozók számának korlátozását előíró megkötések megszegése esetére vezettem be az $UMin_{jk}$, illetve az $OMax_{jk}$ változókat minden egyes ilyen megkötéshez. A célfüggvényben ezek az igény paraméterei közt lévő meghatározott súlyokkal jelennek meg. A megkötés kifejtésében látható, hogy a felírt kényszerek nem biztosítják, hogy ezek a változók ténylegesen azokat az értékeket veszik fel, amelyekkel a célfüggvény számolhat. Szerencsére nincs negatív súly (negatív büntetőpont), és ezen változók pontosan egy darab kényszerben szerepelnek, Így további függőségük nincs. Mivel értékkészletük nemnegatív, ezért pozitív koefficiensükkel együtt továbbra is nemnegatív értékkészletük lesz. Ugyanakkor a célfüggvény minimalizálandó, a koefficienst pedig konstans, így az $UMin_{jk}$, illetve az $OMax_{jk}$ biztosan a legkisebb értéket veszik fel.

5 Az algoritmusok összehasonlítása

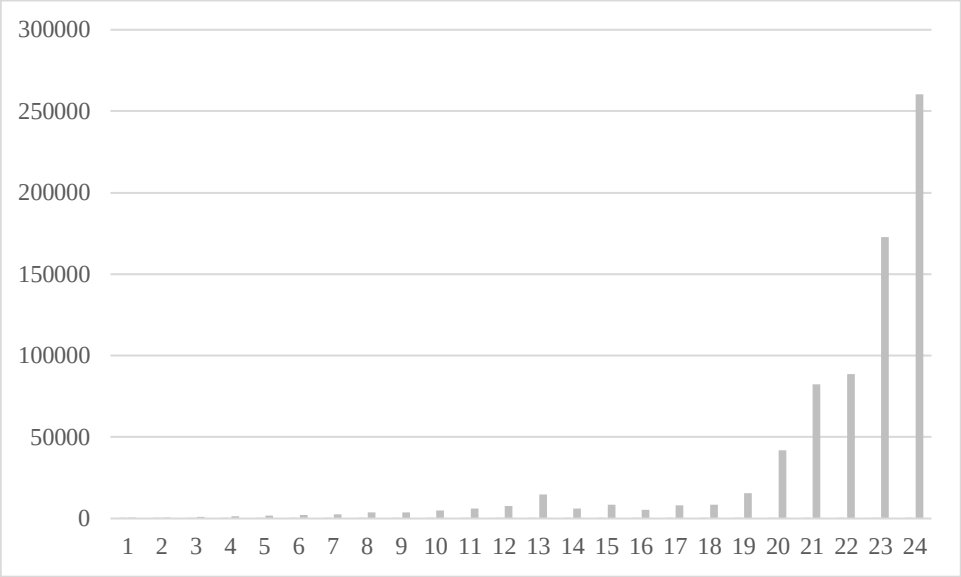
Ebben a fejezetben összehasonlítjuk a két beosztástervező algoritmust a Scheduling Benchmarks szabályai szerint elért pontszámok, illetve a végrehajtási idő alapján, illetve értékeljük őket egyéni szempontok alapján.

5.1 IP Solver értékelése

Az IP megoldás eredményei jól mutatják a módszer előnyeit, és hátrányait is. A legnagyobb előnye az, hogy minden esetben optimális eredményt ad. Ezt jól mutatja, hogy a Scheduling Benchmarks első tesztesetére matematikailag bizonyított, hogy nem adható jobb megoldás a 607-es büntetőpont értéknél. Ezt a megoldásunk elérte.

Ugyanakkor nagy hátránya az IP megoldásoknak, hogy amint a változók vagy a megkötések száma túl nagy lesz, a feladat a legtöbb esetben rendkívül magas futási időt eredményez (ha egyáltalán a gyakorlatban lefut). Ez alól persze vannak kivételek, amikor a feladat LP relaxációja megfelelő megoldást nyújt. Esetünkben nem ez a helyzet. Az első teszteset kivételével egyik teszteset se futott le egy napon belül.

Az IP feladatba felvett kényszereink száma a tesztesetek száma szerint:

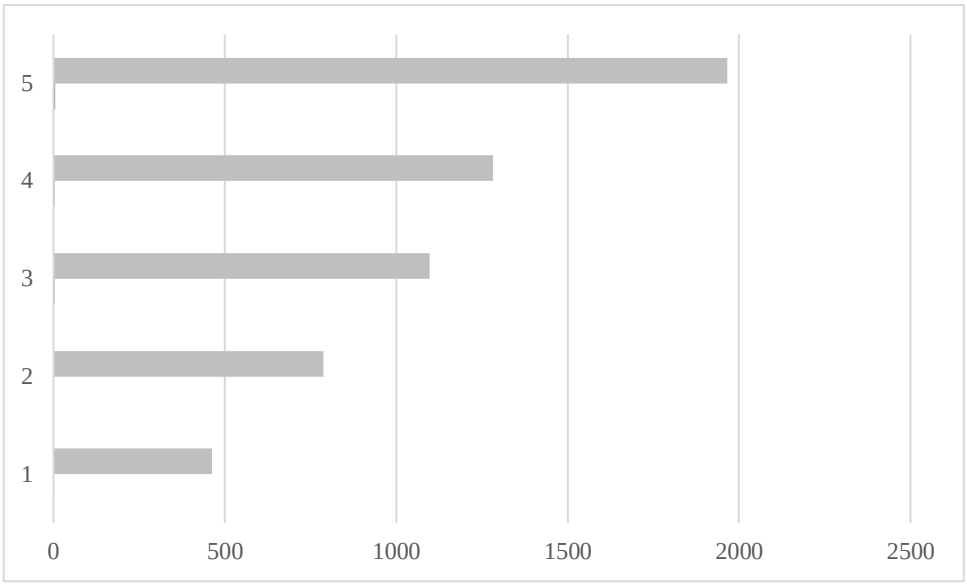


6. ábra: az IP feladatban felvett kényszerek száma tesztesetek száma szerint

Jól látható, hogy a kezdeti pár tesztesthez képest igen nagy ugrást jelent például a 13-as teszteset a maga 14899 kényszerével. Összehasonlítva az első tesztesethez számunkra elegendő volt 462 kényszer. Az egy zavaró tényező lehet, hogy a tesztesetek

különböző bementi paraméter eloszlásai miatt nem igaz az, hogy a nagyobb teszteset számhoz több kényszerre lenne szükségünk.

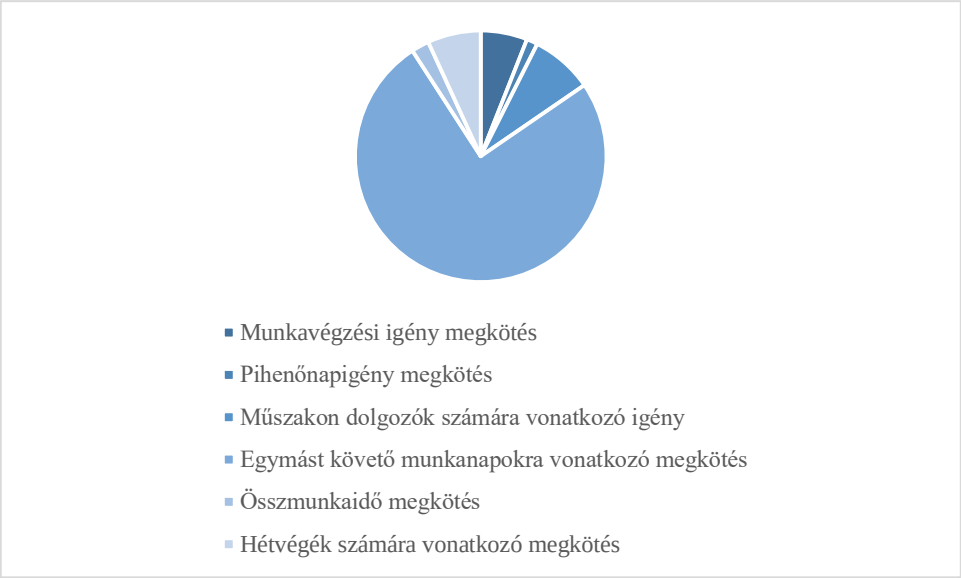
Mivel már a második teszteset problémát jelentett az IP megoldásunknak az első öt tesztesetre tekinteni elegendő. A kényszerek száma ezen tesztesetekben:



7. ábra: kényszerek száma az első öt tesztesetre

Jól látható, hogy ezen teszteseteknél a kényszerek száma közel egyenletesen növekszik.

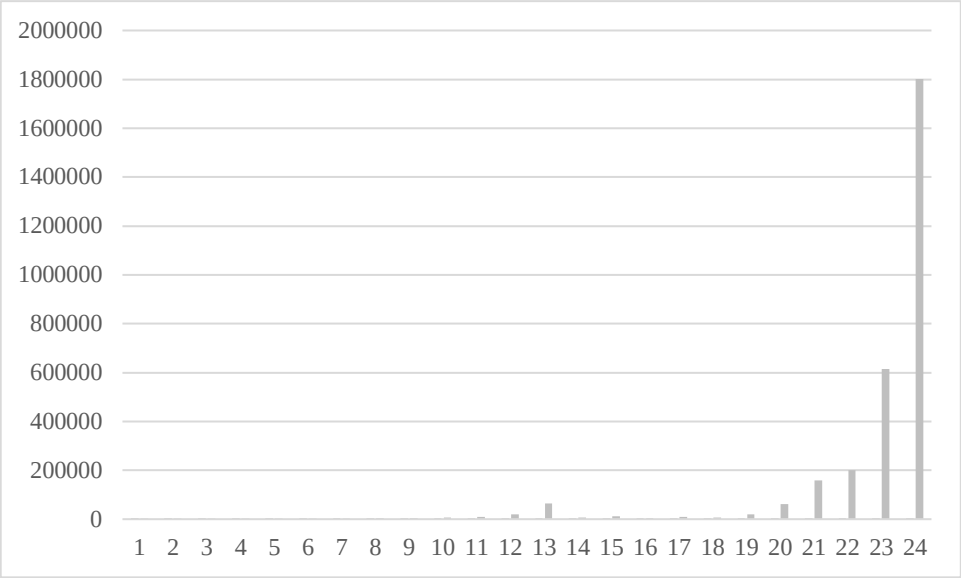
Érdemes még tekinteni, hogy egy megkötéshez hány kényszer tartozik. Ez az első tesztesetre:



8. ábra: megkötésekre lebontva a kényszerek száma az első tesztesetre

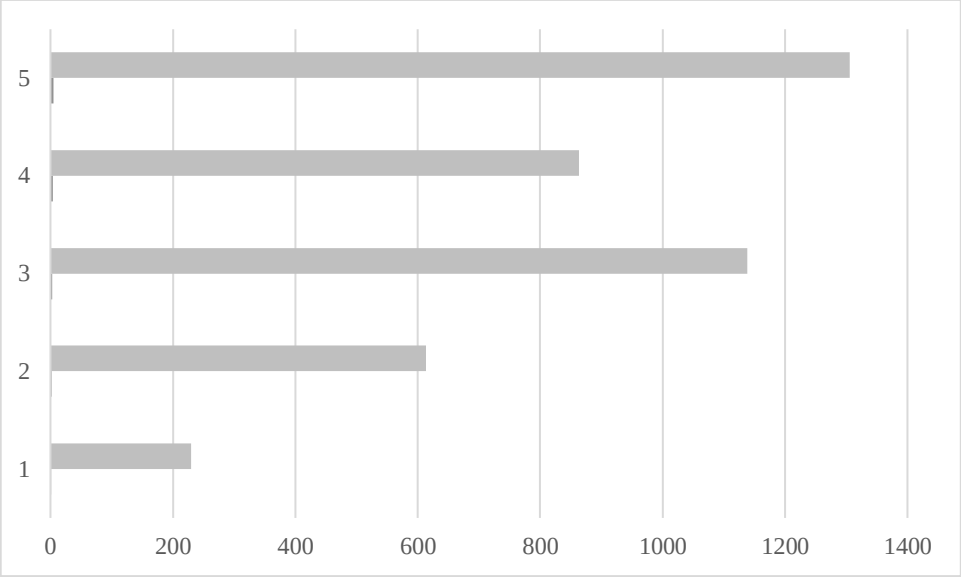
Fontos eredmény viszont, hogy a rögzített munkavégzésre, illetve pihenőnapra vonatkozó kényszereket az IP megoldásunk jól kezeli. Mivel ezek a megkötések nem jelennek meg célfüggvényben (erős kényszerek), ezért egy-egy ilyen felvétele nem módosítja a célfüggvényt, de sokban egyszerűsítheti a számítást. Éppen ezért az input módosítható ilyen jellegű megkötések felvételével és az optimalizációs eljárás helyes marad (amennyiben logikai ellentmondást nem okoz). Ilyen azonban előfordulhat, hogy egy ilyen feltétel felvételével kizártuk az összes olyan beosztást, ami optimum értéket adna. Azonban bizonyos problémás kényszerek akár kiválthatók ily módon (például a hétvégi beosztások előzetes megadásával).

A változók száma alapján is érdemes megfigyelni a kapott eredményeket. Az IP feladatban felvett változóink száma a tesztesetek száma szerint:



9. ábra: az IP feladatban felvett változók száma

Jól látható, hogy a változókra sem igaz, hogy a tesztesetek számával együtt növekedne a számuk. Továbbá itt is megfigyelhető, hogy a tesztesetek többségére viszonylag alacsonyan marad a változóink száma. Mivel a későbbi tesztesetek miatt az ábra nem informatív a kezdeti tesztesetekre, az első öt tesztesetre fókuszáltan is ábrázoljuk a változók számát:



10. ábra: az IP feladatban felvett változók száma az első öt tesztesetre

Itt láthatóan nem egyenletes a változószám növekedés.

5.2 Progresszív algoritmus értékelése

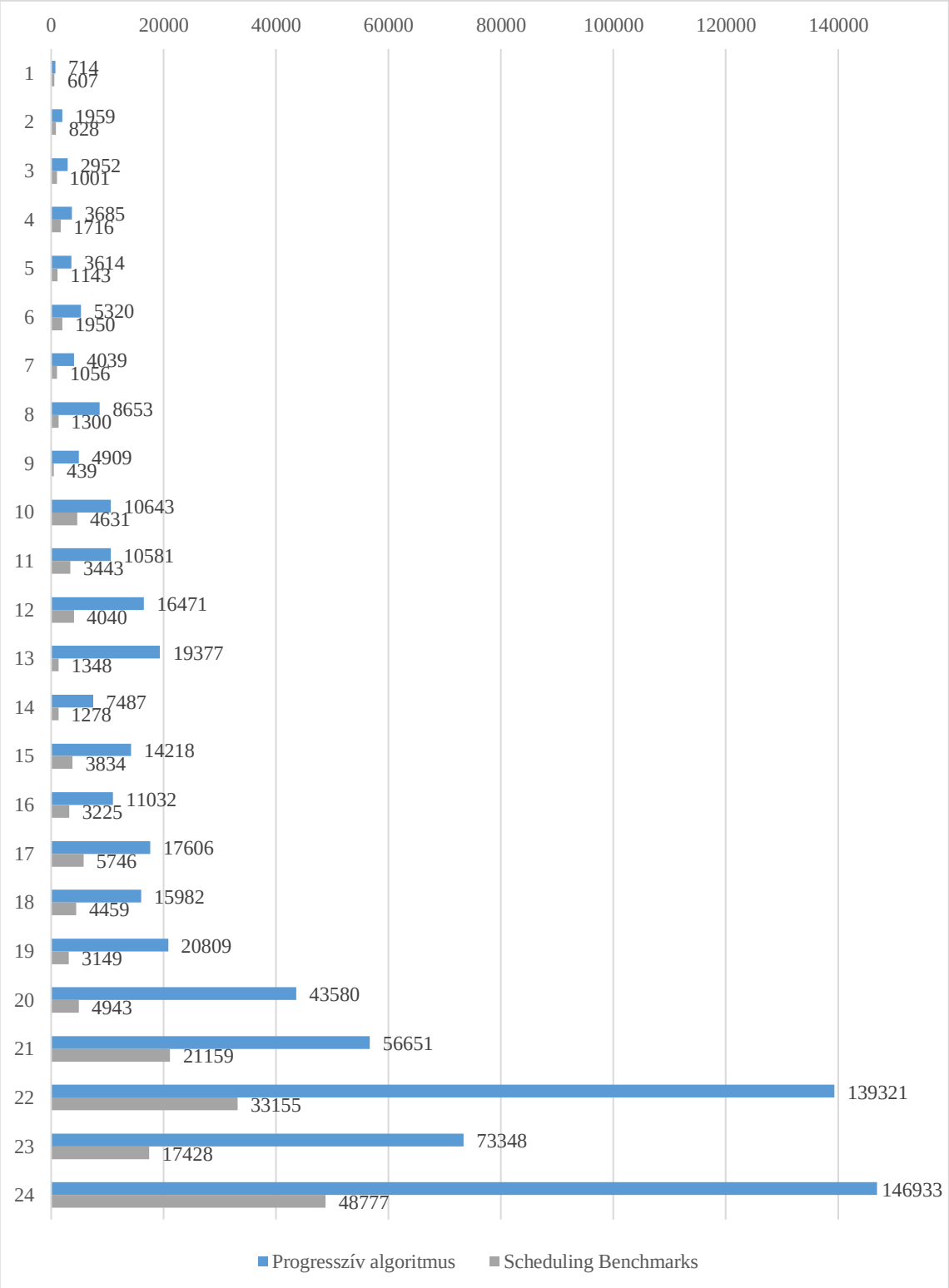
A progresszív algoritmus előnye, hogy nagyon gyorsan ad megoldást, a legnagyobb, 1 éves, 150 dolgozós beosztást is kevesebb, mint fél perc alatt megtervezte. Mindezt úgy, hogy bekapcsolva hagytuk a minimális összmunkaidő eléréséhez szükséges kódrészletet, vagyis minden lehetséges beosztáshoz végigterveztük az adott dolgozó hátralévő időszakát annak ellenőrzésére, hogy elérheti-e még a minimális összmunkaidejét az éppen kérdéses beosztással. A futásidőket a 4. táblázat: A progresszív algoritmus végrehajtási idői mutatja:

4. táblázat: A progresszív algoritmus végrehajtási idői

Teszteset	Végrehajtási idő (ms)
1	69
2	2
3	3
4	3
5	6
6	7
7	7
8	13
9	13
10	25
11	35
12	48

Teszteset	Végrehajtási idő (ms)
13	177
14	29
15	50
16	22
17	51
18	65
19	149
20	858
21	2641
22	3787
23	12434
24	23488

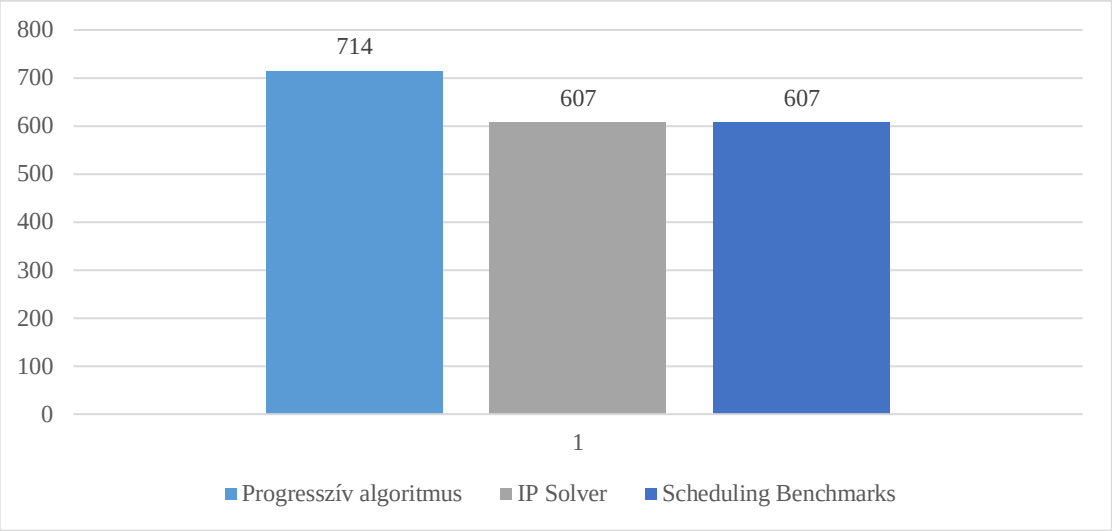
A hátránya viszont, hogy nem ad kielégítő megoldást, nem találja meg minden esetben a legjobb pontszámmal rendelkező végleges beosztást. A Scheduling Benchmarks tesztesetein elért pontszámokat a 11 ábra mutatja be:



11. ábra: A progresszív algoritmus és a Scheudling Benchmarks legjobb elért pontszámai

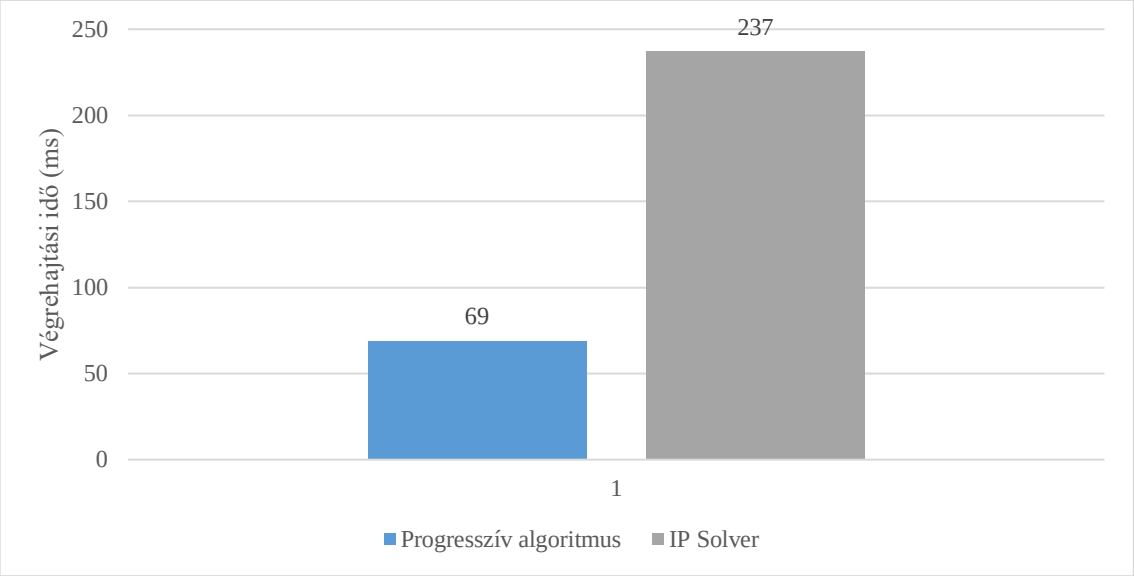
5.3 Összehasonlítás

A Scheduling Benchmarks első tesztesetén elért pontszám alapján az IP Solver maradéktalanul teljesítette a kitűzött feladatot, szemben a progresszív algoritmussal. A 12. ábra hasonlítja össze a két algoritmust és a Scheduling Benchmarks legjobb elért eredményét.



12. ábra: A Scheduling Benchmarks-on elért pontok

A futásidőt összehasonlítva a progresszív algoritmus 3.43-szor gyorsabb az IP Solvernél az első tesztesetre, míg a Scheduling Benchmarks nem tartja nyilván a végrehajtási időket. Az eredményt a 13. ábra mutatja milliszekundumban:



13. ábra: A két algoritmus végrehajtási idejének összehasonlítása az első tesztesetre

6 Összefoglalás

Látható, hogy mind a progresszív megoldásnak, mind az IP megoldásnak van létjogosultsága.

Ha matematikailag optimális eredmény a cél, akkor az IP megoldás a célravezetőbb, ugyanakkor ezzel a megoldással rövid időkorlátok mellett csupán kis létszámú dolgozói állomány egy-két heti beosztására alkalmas.

A progresszív algoritmus esetében bemutattuk, hogyan csökkenti le jelentősen a keresési tér méretét, és az időben előre haladva tervezésnek nem csak előnye, hanem hátránya is van az által, hogy ki tudja zárni az optimális beosztást a keresési térből. Emellett felvetettünk olyan szabályokat is, amiket nem tud hatékonyan betartani. Végül a több körös tervezés bemutatásával láthattuk, hogy a végleges beosztás elkészítésének folyamatában a progresszív algoritmust lehet arra használni, hogy tervezzen egy kiinduló beosztást, amit más megközelítések tovább tudnak javítani.

Bemutattuk, hogy az IP megoldás ténylegesen eléri a matematikai optimumot a Scheduling Benchmarks egy tesztesetén, de beláttuk, hogy a progresszív megoldás futásidőben jelentős előnyt élvez.

Láthatóvá vált, hogy mely feltételek felelősek az IP megoldás futásidejének (és komplexitásának) megnövekedéséért. Továbbá beláttuk, hogy az IP feladat előzetes módosítása lehetséges, hiszen jól kezeli a rögzített munkavégzésre, illetve pihenőnapra vonatkozó erős kényszereket. Ebből kiindulva meghatároztuk, hogy hibrid módszerek használata esetén melyik feltételeket kell más megközelítésre átruházni és mik azok, amiket az IP jól kezel. Ilyen például a hétvégék számára vonatkozó megkötés. Ennek segítségével előzetesen rögzített beosztások vehetők fel az input mellé, drasztikusan csökkentve az egyenlőtlenségrendszer lehetséges megoldáshalmazának méretét.

7 Irodalomjegyzék

[1] E. K. Burke és T. Curtois, „New approaches to nurse rostering benchmark instances," *European Journal of Operational Research,* %1. kötet237, %1. szám1, pp. 71-81, 2014.

[2] *2012. évi I. törvény a munka törvénykönyvéről.*

[3] S. Petrovic, ""You have to get wet to learn how to swim" applied to bridging the gap between research into personnel scheduling and its implementation in practice," *Annals of Operations Research,* pp. 161-179, 2017.

[4] A. Meisels and A. Schaerf, "Modelling and Solving Employee Timetabling Problems," *Annals of Mathematics and Artificial Intelligence,* pp. 41-59, 2003.

[5] J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester és L. De Boeck, „Personnel scheduling: A literature review," *European Journal of Operational Research,* %1. kötet226, %1. szám3, pp. 367-385, 2013.

[6] D. Emir, M. Nysret és W. Felix, „Modeling and solving staff scheduling with partial weighted maxSAT," *Annals of Operations Research,* %1. kötet275, %1. szám1, pp. 79-99, 2019.

[7] R. Erfan, A. Kerem és L. John, „A hybrid Integer Programming and Variable Neighbourhood Search algorithm to solve Nurse Rostering Problems," *European Journal of Operational Research,* %1. kötet258, %1. szám2, pp. 411-423, 2017.

[8] L. Tanguy, B.-M. Odile és P. Damien, „A constraint-based approach for the shift design personnel task scheduling problem with equity," *Computers & Operations Research,* %1. kötet40, %1. szám10, pp. 2450-2465, 2013.

[9] S. Anas Abdoul, D. Laure, L. Corinne és M. Aziz, „Staff scheduling in airport security service," *IFAC Proceedings Volumes,* %1. kötet45, %1. szám6, pp. 1413-1418, 2012.

[10] F. Philip és P. Gerhard, „Personnel Scheduling in Laboratories,” *Practice and Theory of Automated Timetabling IV,* pp. 113-119, 2002.

[11] P. Benedek és K. Bence, „A progressive approach to the employee scheduling problem,” *MultiScience - XXXIII. microCAD International,* 2019.

[12] P. Benedek és K. Bence, „Multiple round employee scheduling with the progressive algorithm,” *Proceedings of the Automation and Applied Computer Science Workshop,* 2019.

[13] C. G. Valicka, D. Garcia, A. Staid, J.-P. Watson, G. Hackebeil, S. Rathinam és L. Ntaimo, „Mixed-integer programming models for optimal constellation scheduling given cloud cover uncertainty,” *European Journal of Operational Research,* %1. kötet275, %1. szám2, pp. 431-445, 2019.

[14] G. Bejarano, D. Rodríguez, J. M. Lemos, M. Vargas és M. G. Ortega, „MINLP-based hybrid strategy for operating mode selection of TES-backed-up refrigeration systems,” *International Journal of Robust and Nonlinear Control,* 2019.

[15] „Shift Scheduling Benchmarks Instances,” [Online]. Available: http://www.schedulingbenchmarks.org/instances1_24.html. [Hozzáférés dátuma: 26 October 2019].

[16] C. Sara, T. T. D. Nguyen, D. C. Patrick, H. Stefaan és S. Andrea, „Second International Nurse Rostering Competition (INRC-II) --- Problem Description and Rules ---,” 2015.

[17] P. J. Fleming, „How not to lie with statistics: the correct way to summarize benchmark results,” *Communications of the ACM,* %1. kötet29, %1. szám3, pp. 218-221, 1986.

[18] P. Franses és G. Post, „Personnel Scheduling in Laboratories,” *Practice and Theory of Automated Timetabling IV,* pp. 113-119, 2003.