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

Hideg Attila

ÓRARENDTERVEZÉS GENETIKUS ALGORITMUS SEGÍTSÉGÉVEL

KONZULENS

Dr. Kővári Bence

BUDAPEST, 2014

Tartalomjegyzék

Összefoglaló

Az órarendtervezés öt évtized óta intenzíven kutatott problémakör, mivel ismert NP-teljes feladatról van szó [1]. Egy órarend elkészítése során korlátozott erőforrások figyelembevételével kell előállítani egy olyan beosztást (helyszínek, oktatók, csoportok és egyéb erőforrások összerendelését), mely megfelel adott feltételeknek. A probléma nem csak a lehetséges megoldások megtalálásáról szól, de azok között az adott mérőszámok szerinti optimum megkereséséről is.

Dolgozatomban a 2007-es International Timetabling Competition által kiírt feladatot oldottam meg, mely egy valós, egyetemi órarend elkészítésén alapul. A verseny lezárása óta számos, az órarendkészítéssel kapcsolatos kutatás vette kiindulási alapul ezt az adathalmazt és igyekeztek különböző módszerekkel minél jobb megoldásokat adni. A nagyméretű keresési tér miatt én genetikus algoritmus segítségével adtam az optimumhoz konvergáló megoldást. Dolgozatomban bemutatom és mérésekkel alátámasztom, miként lehet egy órarendtervező algoritmust a feladathoz igazítani és konkrét módszereket ajánlok a konvergencia gyorsítására.

Az elkészült algoritmusom képes a feltételeknek megfelelő órarendet előállítani, majd a megoldásokat iteratívan javítva az optimumhoz konvergáló eredményt adni. Amennyiben a bemeneti és kimeneti formátumok megfelelnek a fenti verseny formátumának, szinte bármely más órarend-készítési feladatra képes lehetséges megoldást adni.

Abstract

Timetabling problem has been in the focus of a huge researches in the last five decades since it is a well-known NP-complete problem [1]. Timetable creation can be defined as a process of assigning limited resources (rooms, teachers, curricula and other resources) which satisfies a set of constraints. The problem is not only to find a feasible solution but to find an optimal solution using given metrics.

In my paper I am dealing with the 2nd International Timetabling Competition's data sets which are based on a real university's timetable. Since the contest had ended a great deal of research has been done in order to find better solutions using different approaches. Due to the huge search space I have implemented a genetic algorithm which gave me solutions converging to the optimum. In my paper I demonstrate how can an algorithm be adjusted to the given problem and I propose concrete methods which can increase the speed of the convergence and support with measurements.

The proposed algorithm not only can create feasible timetables from the contest's datasets but can iteratively improve the solutions which are converging to the optimum. The algorithm can deal with any other timetabling problem provided that the input and output format is the same as the contest's format mentioned above.

1 Bevezetés

Az órarendtervezés és ütemezés két nagyon hasonló probléma. Mindkettő azzal foglalkozik, hogy minél jobban osszon be limitált számú erőforrásokat, és közben bizonyos kényszereket figyelembe vegyen. Ilyen feladat az, amikor egy irodában a műszakokat megszervezzük, amikor egy áruházban eldöntjük, hogy mikor mennyi raktáros, pénztáros és árufeltöltő kell, vagy amikor egy reptéren a le- és felszálló gépek sorrendjéről döntünk. Bár a problémák mérete más és egy kevésbé jó megoldás hatása is eltérő, közös bennük, hogy a legjobb megoldás megtalálása rendkívül nehéz probléma lehet.

Az órarendtervezés során egy iskola beosztását készítjük el. Ez a beosztás határozza meg, hogy mely embereknek mikor és hol kell lenniük. A probléma maga is igen változatos, és ennek függvényében a nehézsége is az. Egy általános iskolában elég egy hét elkészítése nyolc évfolyam számára minden egyes évben, míg egy egyetemen évi kétszer kell elkészíteni egy ennél komplexebb órarendet számos igény figyelembevételével. Természetesen ki lehet indulni előző éves órarendekből, de évente új feltételek jöhetnek, melyek átszabhatják a teljes folyamatot.

Az órarendek elkészítése sok helyen egy hosszú folyamat, melyet viszont csak ritkán kell elvégezni, évente csupán egyszer vagy kétszer. Az elvégzés azonban sokszor nagyon bonyolult és a megoldása emberi erővel történik. A számítógépek elterjedésével sokat lehet segíteni egy ilyen problémán, hiszen a feladat nagy része automatizálhatóvá válik.

Én az egyetemi órarendtervezéssel fogok foglalkozni, azon belül is a tanterv alapú megközelítéssel. Ennek a lényege az, hogy minden szemeszter elején felállítanak előre meghatározott tanterveket, ahol a tantervek meghatározzák, hogy mely tantárgyakat kell az abba tartozó hallgatóknak felvenniük. A készítés során először meghatározzák ezeket a tanterveket, majd a felmerülő kényszereket összegyűjtik, és végül elkészítik az órarendet. A dolgozat során egy ilyen problémával fogok foglalkozni, adok rá lehetséges megoldásokat, majd értékelem őket, s következtetéseket vonok le a különböző módszerek hatékonyságáról, illetve az egyes megoldások egymásra hatásáról.

2 Az órarend-tervezési probléma

Az órarendtervezés nehézsége a rengeteg különböző feladatból, a nehezen összeegyeztethető követelményekből és a megoldások nagy számából fakad. Az optimum megtalálása NP-teljes probléma [1], melyet az elmúlt öt évtized folyamán nagyon sokan próbáltak minél hatékonyabban megoldani.

2.1 Megoldási módszerek

Ütemezésre és órarendtervezésre már a számítógépek megjelenése előtt is volt szükség. Iskolákban elkészíteni az órarendeket évenként felmerülő feladat volt, melyet sok helyen a mai napig kézzel végeznek. A gyorsítás érdekében egyszerűsítéseket vezettek be a készítés során. Ezek leginkább a kényszerekből indultak ki, például először azokat az órákat osztották be, amelyek egyedi termeket igényeltek, mint a testnevelés. A gyakorlatban gyakran alkalmazott megoldás volt egy már meglévő, például előző éves megoldásból kiindulni.

A számítógépek elterjedésével elkezdték automatizálni az órarendek készítését is. Ezek az algoritmusok az emberi gondolkodást mintázták és ahhoz hasonlóan kezdték el megoldani a problémát. D. de Werra szerint jól ismert problémákra is vissza lehet vezetni ezeket az algoritmusokat, majd azokat megoldani [2]. Cikkében két fajta problémát is bemutat megmutatva, hogyan lehet modellezni ezeket gráfként, illetve hálózati folyamként, majd megoldási javaslatot ad rájuk.

Az évek során nagyon sok kutatás indult ki olyan algoritmusokból, melyeket gyakran és hatékonyan használtak más kombinatorikai optimalizációs problémákra. Ezek az algoritmusok valamilyen heurisztikával igyekeztek minél jobb, de nem feltétlenül optimális megoldást találni. De Werra és A. Hertz [3] foglalkozott metaheurisztikák alkalmazásával, amelyek úgy akarnak jó (esetleg optimális) megoldást adni, hogy lépésenként egy alacsonyabb szintű heurisztikát alkalmaznak. Ezek használata sokszor jobb eredményhez vezetett, mint az egyszerű heurisztikán, vagy az iteratívan javító algoritmuson alapuló megoldások [4]. A metaheurisztikán alapuló algoritmusok nagy részét felhasználták az órarendtervezésben is, többek között genetikus algoritmust [5], hangya kolónia optimalizálást [6], részecske raj alapú optimalizálást [7], tabu keresést [8, 9], szimulált lehűtést [10] vagy ezek kombinációját alkalmazva [11].

2.2 Genetikus algoritmus

Egy igen népszerű metaheurisztikus keresés a genetikus algoritmus is, mely sok optimalizációs problémánál hatékonynak bizonyult. A következőkben röviden összefoglalom a genetikus algoritmus lényegét és bemutatom néhány alkalmazási területét.

2.2.1 Az evolúció mintázása

A genetikus algoritmus elméleti alapjait John H. Holland [12] fektette le, mely alapvetően az evolúciót utánozza. Működési elve, hogy egyedek egy populációját létrehozva iteratívan javítja a megoldásokat. Az evolúció során az egyedek úgy adaptálódtak a környezethez, hogy a génállományukból a lehető legmegfelelőbbeket tartották meg, a feleslegeseket elhagyták. Ezen egyedek nagyobb túlélési eséllyel rendelkeztek, így az új egyedek nagyobb eséllyel kaptak jobb géneket, melyeket aztán továbbörökítve még jobb egyedeket kapunk. Holland az egyedeket kromoszómákként jellemezte, mely az említett génállományt tartalmazta. A környezet a problémát jelenti, a kromoszóma pedig egy lehetséges megoldás az adott problémára, melynek reprezentációja pedig mindig más, az adott problémától függ. Egy úgynevezett fitness fogalmat is bevezet, mely célja, hogy a problémában való „megfelelőséget" számszerűsítse és egy függvény segítségével kiszámolhatóvá tegye. Bevezetésre kerülnek úgynevezett genetikus műveletek is, melyeket a kromoszómákon lehet alkalmazni. A leggyakoribb műveletek a keresztezés, a mutálódás, a szelekció, melyeket én is felhasználok.

A megoldás keresése során egy kiinduló populációval dolgozunk, mely célja, hogy az összes eddig megoldásunkat összefogja. A populációban különböző módszerekkel kiválasztott egyedeket keresztezünk. A keresztezés során a két szülőből egy, vagy két gyermek egyed jön létre, melynek tulajdonságai ideális esetben jobbak lesznek, mint a szülő tulajdonságai. A mutáció során egy egyedet változtatunk meg, mely mértéke implementációfüggő: lehet egy-egy gént, de akár a teljes egyedet megváltoztatjuk. A keletkező új egyedeket valamilyen módszerrel beillesztjük a populációba.

2.2.2 Genetikus algoritmus alkalmazása

A genetikus algoritmust akkor szokták alkalmazni, ha a hagyományos módszerekkel történő megközelítés nem olyan egyszerű, esetleg nem lehetséges. Használata egyszerű feladatokra nem célravezető, mivel futási ideje és erőforrásigénye is nagyobb lesz, nem mérhető feladatokra pedig egyáltalán nem is lehetséges. Nem mérhető feladatnak számítanak az olyan feladatok, ahol például az eredmény egy egyszerű igaz/hamis, de azon belül nem különböztetjük meg őket. Ebben az esetben a genetikus algoritmus nem több, mint egyszerű véletlen megoldások előállítása, mindenféle heurisztika nélkül.

Néhány alkalmazási terület:

- Automatizált kereskedőszoftverek stratégiájának fejlesztése
- Elektromos hálózatok tervezése
- Mobilhálózatok infrastruktúrájának optimalizálása
- Útvonaltervezés, forgalmi irányítás

2.2.3 A genetikus operátorok szerepe a keresésben

A keresési tér bejárása során két fogalmat különböztetnek meg: exploration és exploitation. Mindkét fogalom azt jelenti, hogy újabb megoldásokat keresünk, különbség a módszerükben van. Az exploration során kísérletezéssel gyorsan bejárjuk a keresési teret ígéretes megoldások után kutatva, sokszor a meglévő megoldásoktól távol álló lehetőségeket is megpróbálva. Ez alapvetően egy globális keresést jelent, melynek során a vizsgált elemek nagyon változatosak lesznek. Az exploitation során csupán egy kisebb részét járjuk be a keresési térnek. Ez a tér egy ígéretes megoldásból indul ki, és a keresés annak környékén történik, finomítva azt.

A kettő közötti egyensúly megtalálása nagyon fontos. Amennyiben kizárólag az előbbit alkalmazzuk a megfelelő megoldásoktól akár nagyon távol is kerülhetünk, megnövelve a kereséshez szükséges időt, míg kizárólag exploitation módszerrel dolgozva könnyen rekedhetünk lokális optimumban.

A genetikus operátorok közül a mutáció és a keresztezés megfeleltethető az exploration módszernek. A keresztezés során az egyedek génjei a szülőktől öröklődnek, a mutáció során pedig egy-egy gént változtatunk meg. A mutáció szerepe az optimalizálás

során kiveszett tulajdonságok ismételt bevezetése. Például ha a populáció összes kromoszómájának az egyik tulajdonsága megegyezik, akkor a keresztezés segítségével az összes újonnan keletkezett egyedben ugyanez a tulajdonság lesz. Ha azonban az optimális megoldásokban nem ez szerepel, akkor keresztezés segítségével nem érhetjük el azokat.

Az exploitation alapja viszont a szelekció, melynek alapvetően két szerepe van: a populáció számának limitálása, és a megoldások halmazának szűkítése. Előbbi egyértelműen fontos, ugyanis több tíz- vagy százezer egyed hatalmas teljesítményromlást eredményezhet, sőt akár a memóriát is telítheti. A megoldások szűkítése azért is fontos, mivel ekkor szűrjük ki a potenciális megoldások közül a rosszabbakat.

A fentiekből következik, hogy a keresztezés, a mutáció és szelekció nem megfelelő aránya esetén nehezen találhatjuk meg a megoldást. A túl kicsi arányú szelekció esetén nagyon sokszor a keresés nem megfelelő megoldások irányába megy, a túlzott szelekció esetén pedig lokális optimumban ragadhatunk.

3 A felhasznált probléma

Az órarend-tervezéssel kapcsolatban már több versenyt is rendeztek, melyeknek célja volt, hogy egyre bonyolultabb problémákon limitált idő alatt adjanak minél jobb megoldást a kiadott leírásnak megfelelő adathalmazra. Munkám során a második, 2007-es International Timetabling Competition adatsoraiból indultam ki. Ezeket az adatsorokat a verseny résztvevőinek adták ki előre, hogy a saját algoritmusaikat tesztelhessék még az éles leadás előtt. A verseny során három különböző fajta órarend-tervezési problémát oldottak meg az indulók. Ezek közül én a tanterv alapú megközelítést használtam.

3.1.1 Probléma leírása

A tantervalapú órarend-tervezés lényege [13], hogy előre definiált tanterv mentén kell összeállítani az órarendet. A megoldás időpont-óra-terem hármasok halmaza lesz. A probléma pedig az alábbi fogalmakat használja:

Megvalósítható órarend: olyan órarend, mely nem sért kemény kényszert.

Tanterv: a feladat alapját képző tantervek meghatározott tárgyakat foglalnak magukban. A közös tantervben szereplő tárgyakat ugyanazon hallgatók veszik fel és ezek nem ütközhetnek.

Időpont: a hetet napokra, a napokat időközökre bontjuk és ezen időközök összességét lehet felhasználni, mint időpontokat. Egy hét tipikusan öt-hat napból áll, és minden nap ugyanannyi időközre lesz bontva.

Tárgy és tanár: minden tárgy megadott számú órából áll, melyeket egy előre megadott oktató tanít és megadott számú diák hallgat. Ezeket az órákat kell majd különböző időpontokra beosztani. Egy tárgy több tantervben is szerepelhet.

Terem: a termekben kerülnek az órák megtartásra. Minden teremnek van egy egyedi azonosítója és kapacitása.

3.1.2 Kemény kényszerek

A kemény kényszerek határozzák meg, hogy az órarend megvalósítható-e. Elsődleges célunk, hogy az elkészült órarend ne sértse meg őket. A probléma az alábbi kemény kényszereket definiálja:

Órák ütemezése: minden tárgy minden óráját be kell ütemezni, egy tárgy órái nem kerülhetnek azonos időpontban megtartásra.

Terem fogaltsága: egy adott időpontban, egy teremben csak egy órát lehet megtartani.

Ütközés: azonos tanrendbe tartozó, valamint az azonos oktató által tanított órákat nem lehet egy időpontba osztani.

Rendelkezésre állás: bizonyos időpontokban adott tanárok nem érnek rá, így ezekre az időpontokra nem lehet őket beosztani.

3.1.3 Laza kényszerek

A laza kényszerek büntetőpontok formájában fognak megjelenni a végeredményben. A célunk, hogy a bűntetőpontok száma minimális legyen. Ideális esetben ez nulla, de a kiadott problémák közt van olyan, amire nincs ilyen megoldás. A következő laza kényszereket kell figyelembe venni:

Teremkapacitás: minden terembe legfeljebb annyi diákot oszthatunk be, amennyi annak a kapacitása. Efölött minden diák egy büntetőpontot ad a megoldáshoz.

Minimum oktatott napok: minden tárgynál adott, hogy hány napon kell oktatni, melynek célja, hogy elosztva legyenek leadva a hét folyamán. Minden egyes nap, amennyivel kevesebben szerepel az adott tárgy, öt büntetőpontot jelent.

Elszigetelt órák: egy tantervben az órák egymás után kerüljenek leadásra. Minden egyes óra, mely előtt és után sincs óra két büntetőpontot jelent.

Teremhasználat: egy tárgy minden órája ugyanazon terembe kerüljön. Minden különböző terem használata egy büntetőpontot ad hozzá.

4 Tesztelés leírása

4.1 Tesztkörnyezet

Az elkészült algoritmus tesztelése a már említett 2007-es International Timetabling Competition publikus adatsoraival történik [14]. A verseny szervezői biztosítottak egy eszközt az indulóknak mely segítségével meghatározható, hogy mennyi ideig dolgozhat az algoritmus a teszteset megoldásán. Az általam használt gép egy kétmagos, 2,5 GHz-es Intel Core i5-ös processzorral és 8 GB RAM-mal rendelkező laptop volt, melyen Windows 8.1 operációs rendszer futott. Ezen a konfiguráción 286 másodpercig dolgozhatott a gép. A megoldás előállításához egy saját keretrendszert valósítottam meg C# nyelven, majd a verseny szervezői által kiadott c++ forráskód segítségével készült kiértékelővel bizonyosodtam meg a költségéről. A bemeneti formátumot az A függelék, a kimeneti formátumot a B függelék, a kiértékelő által adott kimenetet pedig a C függelék írja le.

4.2 Tesztelés futtatása

A tesztelés során a verseny első feladatát oldottam meg. A feladat az volt, hogy az adott időkereten belül olyan órarendet készítsünk, mely kemény kényszerekből eredő költsége (továbbiakban kemény költsége) 0, és a laza kényszerekből eredő költsége (továbbiakban laza költsége) minél kisebb. Ennek érdekében az algoritmus futását az időlimit leteltével leállítottam és kiértékeltem az eredményt. A publikus tesztesetek közül az elsőt használtam fel, mely segítségével megvizsgáltam az algoritmus változtatásainak a hatását. A teszteset tulajdonságai:

Órák száma: 126

Termek száma: 6

Időpontok száma: 30

5 Órarendtervező algoritmus

Ebben a fejezetben megvizsgálom, hogy milyen hatást gyakorolhat a genetikus algoritmus különböző komponenseinek megválasztása a végső futás hatékonyságára. Referenciaként a véletlenszerű beosztást használva különböző módszereket adok a hatékonyság javítására.

5.1 Véletlenszerű órarend

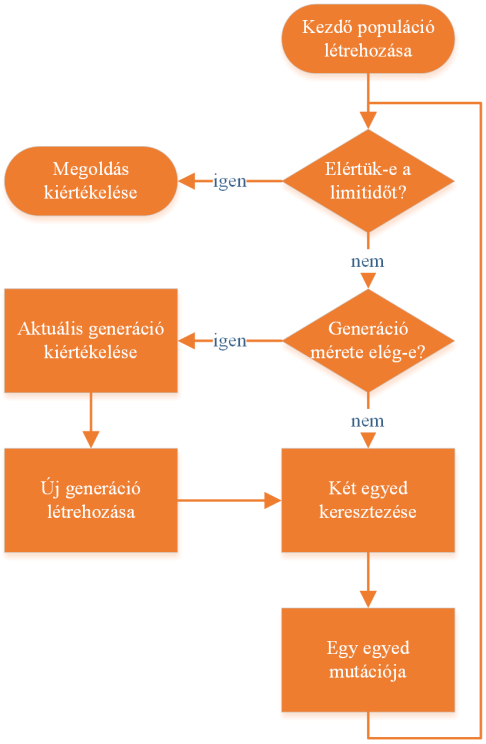
Már ez első tesztesetnél is látható, hogy a lehetséges megoldások halmaza hatalmas: minden egyes órához hozzá kellett rendelni 6 terem és 30 időszelet valamelyikét, ami 126 óra esetén $(6 * 30)^{126}$ lehetséges megoldást jelent, ami 10^{284} nagyságrendű. Ez persze tartalmazza az összes megoldást, azokat is, amik nem adnak megvalósítható órarendet.

Az első megoldás a véletlenszerű eredmények készítése volt. Az összes órához vettem egy véletlenszerűen választott termet és időpontot, majd ezt addig ismételtem, amíg le nem telt az időkorlát. A legjobb eredmény az alábbi volt: 87,1519. Ez azt jelenti, hogy 87 kemény költsége és 1519 laza költsége volt.

Az algoritmus messze nem ad jó eredményt, egyszerű finomításokat vezetek be: ne tegyünk ugyanolyan órát egy időpontra, és vizsgáljuk meg, hogy egy időpontra be tudunk-e tenni egy órát, és ha nem, csak akkor tegyük oda, ha már máshova sem lehet. Ezen kívül, ha lehet, egy tárgy óráit egy terembe osszunk be. Ezzel a megoldással az alábbi fitness értéket kaptam: 67,2357.

5.2 Az algoritmus ismertetése

A genetikus algoritmus konkrét implementációja igen változatos lehet, de a lépések nagyjából hasonlóak. Az általam létrehozott algoritmus futása a 1. ábrán látható. A futtatás előtt a populációt inicializálom kezdőelemekkel. Fontos, hogy ezek az elemek minél változatosabbak legyenek, mivel a túl hasonló tulajdonságú egyedek lecsökkentik a rajtuk keresztül bejárható keresési teret. A kezdő populációból kiindulva újabb generációkat hozok létre, míg el nem érjük a megállási feltételt. Ez a feltétel nagyon sokféle lehet, de az én esetemben adott: a verseny során megadott időlimit elérése.



1. ábra: Genetikus algoritmus általános szemléltetése

Minden újabb generáció elkészítése a következő módon történik: kiválasztom a szelekciós operátor segítségével, hogy melyik két szülőegyedet keresztezem, majd létrehozok két gyermeket és ezeket beleteszem az új generációba. Ezután kiválasztunk egy egyedet, majd a mutációt rajta végrehajtva belehelyezzük az új generációba. Az új generáció bővítését mindaddig folytatjuk, amíg el nem érjük a populáció méretét. Ezután az új generáció fogja alkotni a populációt és abból hozunk létre egy új generációt.

5.3 Kromoszóma

Az általam definiált kromoszóma egy kész órarendet tárol. A következőkben ismertetem a létrehozott kromoszóma legfontosabb tulajdonságait.

5.3.1 Fitness

A kromoszóma legfontosabb tulajdonsága a fitness értéke. Ez egy függvény, mely megadja, hogy mennyire életképes a kromoszóma az adott problémában. Ennek a célfüggvénynek az értékét kell minimalizálni vagy maximalizálni a problémától függően. Általános esetben a fitness képlete:

$$f = \sum f(x_i)$$

Ahol x_i az i. gén értéke, f pedig a gén értékéből számított költségfüggvény.

Órarendtervezésnél a célfüggvény, amit minimalizálni kell, a kényszerekből eredő költségek összege lesz. Ezeket a költségeket sok esetben, ahogy esetünkben is, besorolják puha és laza kényszerek kategóriájába. Az így kapott képlet a következő:

$$f = \sum h(x_i) * C + s(x_i)$$

Ahol *h* a gén kemény, *s* pedig a laza kényszerből adódó költségfüggvény. Mivel az elsődleges cél az, hogy a kemény költség nélküli órarendet találjunk, így a *h* függvényt súlyozzuk egy *C* konstanssal, melyet nagyobbra választunk, mint a lágy kénszerek által kiadott összköltség.

Mivel az én esetemben a legtöbb kényszer megsértése nem csupán egyetlen géntől függ, így a kiszámolás a fentinél egy kicsit bonyolultabb. Például az elkülönített órák meghatározásához tudnunk kell, hogy előtte és utána sincs-e óra. Másik fajtája ennek, amikor egy tárgyhoz tartozó összes gén sért egy kényszert, például ha nincs elég napon óra tartva belőle. Ekkor az összes óra, mely a tárgyhoz tartozik szerepet játszik ennek a költségnek a megformálásában. Annak érdekében, hogy a fenti képletet használni tudjam, illetve könnyebben meg tudjam határozni a büntetőponttal rendelkező géneket, miután az egyes költségeket kiszámoltam, az ahhoz hozzájáruló gének egyenlő súllyal megkapják a ráeső részt. Ezek után használhatom a fenti képletet.

C konstans választása: a konstans értékét 10000-ben határozom meg. Ez egy felső korlát *s* függvényre, ugyanis a lágy kényszerek költsége egyik tesztesetben sem éri el ezt.

h függvény részletezése:

Órák ütemezése: alapvetően minden órához létrehozok egy gént. Ha ez nem kerül beütemezése, vagy olyan helyre ütemezzük, ahol ugyanilyen óra van, eggyel növeli a kemény költséget.

Terem fogaltsága: azok a gének, melyek egy időpontban vannak, és ugyanazt a termet használják génenként $\frac{n-1}{n}$. ponttal növelik a kemény költséget, ahol n ezen gének száma.

Ütközés: azok a gének, melyek órái azonos tanrendbe tartoznak, vagy azonos oktató által tanítottak és megegyező időpontban vannak génenként $\frac{n-1}{n}$. ponttal növelik a kemény költséget, ahol n ezen gének száma.

Rendelkezésre állás: azok a gének, melyek óráit tanító tanár az adott időpontban nem elérhető eggyel növelik a kemény költséget.

s függvény részletezése:

Teremkapacitás: minden gén, mely olyan órát tartalmaz, ami több hallgatót tartalmaz, mint a génhez rendet terem kapacitása, annyival növeli a laza költséget, amennyivel túllépjük a kapacitást.

Minimum oktatott napok: minden gén, mely olyan tárgyhoz tartozó órát tartalmaz, ami nem szerepel annyi napon, mint amennyit a tárgy megkövetel, növeli a laza költséget génenként $\frac{M-s}{n}$. ponttal növelik a laza költséget, ahol n ezen gének száma, M a tárgy által megkövetelt napok száma, s pedig a tárgyhoz tartozó napok száma.

Elszigetelt órák: minden gén, mely olyan tárgyhoz tartozó órát tartalmaz, mely előtt és után sincs egy tantervben lévő tárgyhoz tartozó óra, minden egyes ilyen tantervenként eggyel növeli a laza költséget.

Teremhasználat: minden gén, mely egy olyan tárgyhoz tartozó órát tartalmaz, ami több termet is igénybe vesz génenként $\frac{s-1}{n}$. ponttal növelik a laza költséget, ahol n ezen gének száma, s pedig a tárgy által igénybevett termek száma.

Annak érdekében, hogy a fitness értéke egyből tükrözze a költségek eloszlását, osztom C-vel a kapott képletet. Így a fitness értékének egészrésze megadja a megoldás kemény költségét, a törtrésze pedig a lágy költségét. Egy egynél kisebb fitness érték megvalósítható órarendet fog jelenteni.

5.3.2 Reprezentáció

A kromoszóma helyes felépítésének, vagyis a gének struktúrájának helyes megválasztása kritikus kérdés. Fontos, hogy a kromoszómából a választ viszonylag könnyen elő lehessen állítani, a megoldást könnyen ki lehessen értékelni és a módosítás is gyors legyen. Amikor kiválasztjuk, hogy melyik reprezentációt fogjuk használni, el kell döntenünk, hogy a három műveletből melyikre szeretnénk optimalizálni. Mivel a megoldás előállítása csak a végén történik, így két módszerrel dolgozom: az első a kiértékelést gyorsítja, a másik pedig a módosítást.

5.3.2.1 Tanterv alapú reprezentáció

A kromoszómában található gének egy-egy óra-időpont-terem hármasból állnak. A géneket úgy tárolom, ami a legközelebb áll az emberi gondolkodásmódhoz: tantervekre bontva táblázatos formában. Az 2. ábrán jól látszik, hogy a kiértékelés miért gyors: a táblázatos formátum minden egyes tantervhez egy kétdimenziós tömböt jelent. A kiértékelésnél sorra veszem a tanterveket, majd azon belül a kétdimenziós tömböket. Ez azt jelenti, hogy három egymásba ágyazott ciklust kell végigszámolni ahhoz, hogy megtudjuk a kényszerek költségét.

A módszer hátránya azonban a táblázat karbantartása. Mivel vannak átfedések a tantervek között, így amikor egy olyan gént akarunk megváltoztatni, melyben található óra több tantervben is szerepel, akkor mindegyiknél módosítani kell. Ha ez a módosítás ráadásul egy időpont megváltoztatása, akkor a táblázatokban is arrébb kell tenni a megfelelő helyre. A 2. ábrán látható kromoszóma esetén például az első időpontban lévő óra közös, annak áthelyezése esetén mindkét tanterv tömbjét módosítani kell.

$Curriculum_1$			
	Day_1	Day_2	Day_3
$Period_1$	$Lecture_1$ - $Room_2$		
$Period_2$	$Lecture_2$ - $Room_4$		$Lecture_4$ - $Room_1$
$Curriculum_2$			
	Day_1	Day_2	Day_3
$Period_1$	$Lecture_1$ - $Room_2$		
$Period_2$		$Lecture_3$ - $Room_3$	

2. ábra: Egy tanterv alapú kromoszóma

5.3.2.2 Esemény alapú reprezentáció

A gének maguk ebben az esetben is egy óra-időpont-terem hármasból állnak. Ezen hármasok összessége fog kiadni egy kész órarendet. A géneket egy listában tárolom, melyeket az órával címzek meg. Ennek a reprezentációnak az előnye, hogy az egyes elemek változtatásánál nem kell karbantartani másokat, így jelentősen meggyorsítja a keresztezés és mutálódás műveleteket.

$Lecture_1$	$Lecture_2$	$Lecture_3$	$Lecture_4$	$Lecture_5$
$Room_1$	$Room_2$	$Room_3$	$Room_1$	$Room_2$
$Timeslot_3$	$Timeslot_4$	$Timeslot_2$	$Timeslot_5$	$Timeslot_4$

3. ábra: Egy esemény alapú kromoszóma

A listát úgy készítem el, hogy az összes tárgyhoz tartozó órát egymás után a listába helyezem, majd ezekhez választunk ki egy termet és egy időpontot a 3. ábrán látható módon.

Mivel a cél a gének gyors módosítása a változtatáskor nem ellenőrzöm, hogy az adott módosítás megfeleljen kényszereknek. Ennek következtében azt is megengedem, hogy egy időpontra bármennyi óra kerüljön, és amennyiben kényszert sértenek, a kiértékelés során a költséget fogják növelni. Ennek a legnagyobb hátránya, hogy nem tudok kizárni egy kemény kényszert sem a módosítások során. Mivel az összes óra szerepel a listában, elsőre úgy tűnik, hogy azt a kényszert nem sértjük, ám a kiértékelő sajnos nem így működik. Amennyiben egy tárgyhoz két óra ugyanahhoz az időponthoz kerül, azt a kiértékelő ugyanannak a sornak veszi és eldobja az utóbbit. Ennek következtében az nem járul hozzá ugyan más büntetőponthoz, de egy kemény kényszert sért.

A módosításra optimalizálás hátránya a kiértékelésnél megnövekedett számítás idő. Annak érdekében, hogy ezt is gyorsítsuk, kétdimenziós segédmátrixokat vezetek be, melyek egy részét kiértékelés előtt, a többit pedig a kiértékelés során számolom ki. A következő segédmátrixokat vezetem be:

Összeférhetetlenségi mátrix: azon órákat tartja számon, melyek nem kerülhetnek azonos időpontba, vagy azért mert egy tantervben vannak, vagy azért mert közös az oktatójuk. Elég egyszer kiszámolni, mert nem változik

Rendelkezésre állási mátrix: azon órákat és időpontokat tartja számon, melyek nem kerülhetnek egy génbe, mivel az oktató nem ér rá ekkor. Elég egyszer kiszámolni, mert nem változik.

Teremhasználat: az időpontok és termek kapcsolatát reprezentáló mátrix. Minden cellájába az kerül, hogy hány órát osztottunk adott időpontra és terembe. Minden egyes új génnél külön kell kiszámolni.

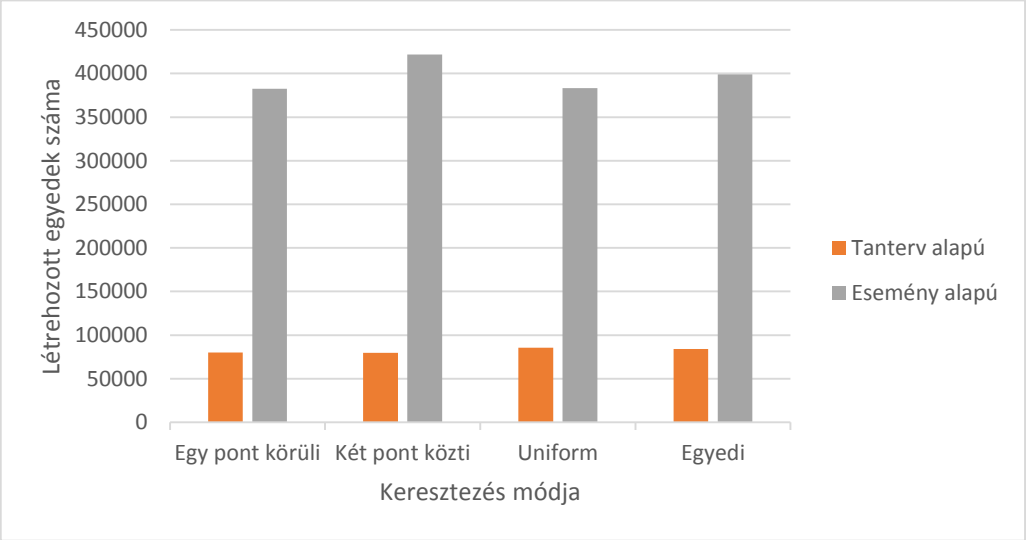
Naphasználat: az egyes tárgyakhoz tartom nyilván, hogy mely napokon van hozzá tartozó óra tartva. Minden egyes új génnél külön ki kell számolni.

Tanterv: az egyes tantervekhez tartom nyilván, hogy mely időpontokban vannak óráik, így könnyebb meghatározni az izolált órák számát. Minden egyes új génnél külön ki kell számolni.

5.3.3 Reprezentációk tesztelése

A reprezentációk teszteléséhez létrehoztam egy 150 fős populációt, és azon futattam az algoritmust, az alábbi paraméterekkel:

- Szelekció: az 5.6.1-es pontban ismertetett véletlenszerű kiválasztás
- Mutáció: az 5.5.1.1-es pontban bemutatott véletlenszerű mutáció
- Keresztezés: az 5.4-es pontban bemutatottak mindegyikét kipróbáltam



4. ábra: Egyedszám a különböző megoldásokkal

A 4. ábrán a függőleges tengelyen látható azon egyedek száma, melyet mutációval vagy keresztezéssel hoztunk létre, vízszintes tengelyen pedig a módszerek találhatóak. Megfigyelhető, hogy a tanterv alapú reprezentáció minden esetben alulmaradt az eseményalapúhoz képest. A grafikonon Ennek az oka, hogy a keresztezés során keletkező többletmunka mértéke sokkal nagyobb volt, mint amit a fitness számolásánál nyertünk, így az egyedszám körülbelül az ötöde lett, mint a másik esetben. Mivel a genetikus algoritmus nagyban épít arra, hogy minél több egyedet hozzon létre, ez visszavetette a kapott eredményeket is.

A Visual Studio saját diagnosztikája segítségével megmértem, hogy mennyi CPU-időt tölt a program futása során az egyes függvények kiszámolásával. Nem meglepő módon a tanterv alapú reprezentáció esetén a keresztezés jóval több időt vett igénybe, körülbelül ugyanannyit, mint a fitness kiértékelése. Ezzel szemben az esemény alapú esetében a keresztezés negyed annyi ideig tartott. A véletlenszerű mutáció mindkét esetben elhanyagolható mennyiségű időt vett el.

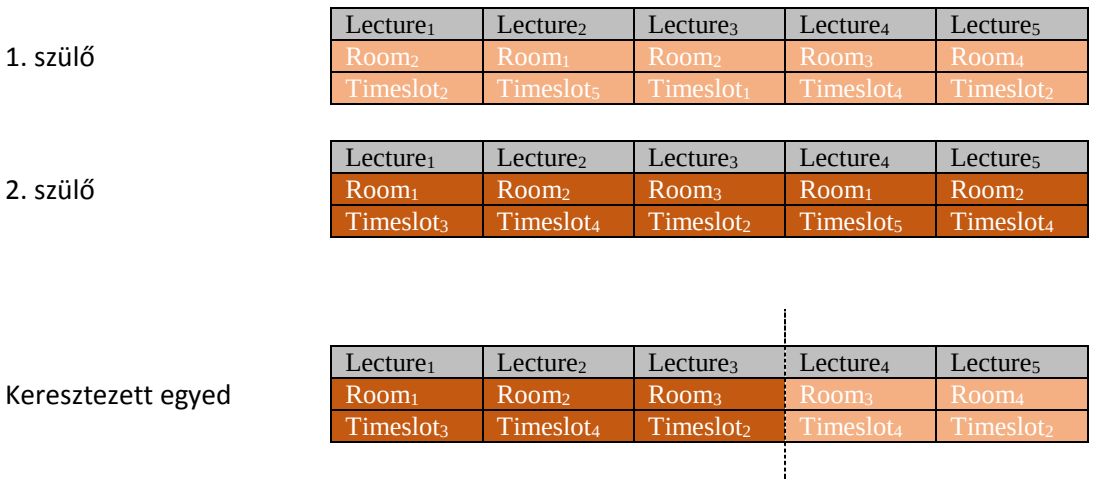
5.4 Keresztezés

Az általam vizsgált keresztezések olyan rekombinációs folyamatok, melyek során két szülő egyedből két gyermek jön létre. Mindkét gyermek egyed létrehozása ugyanúgy történik, a két gyermek célja, hogy csökkentsük annak az esélyét, hogy ha két potenciálisan jó szülőből választunk, akkor a rossz választás miatt rosszabb gyermek egyedet kapunk. A következőkben bemutatok négy keresztezési módszert, melyeket az órarendtervezés során használok:

- Egy pont körüli keresztezés
- Két pont közti keresztezés
- Uniform keresztezés
- Egyéni keresztezés

5.4.1 Egy pont körüli keresztezés

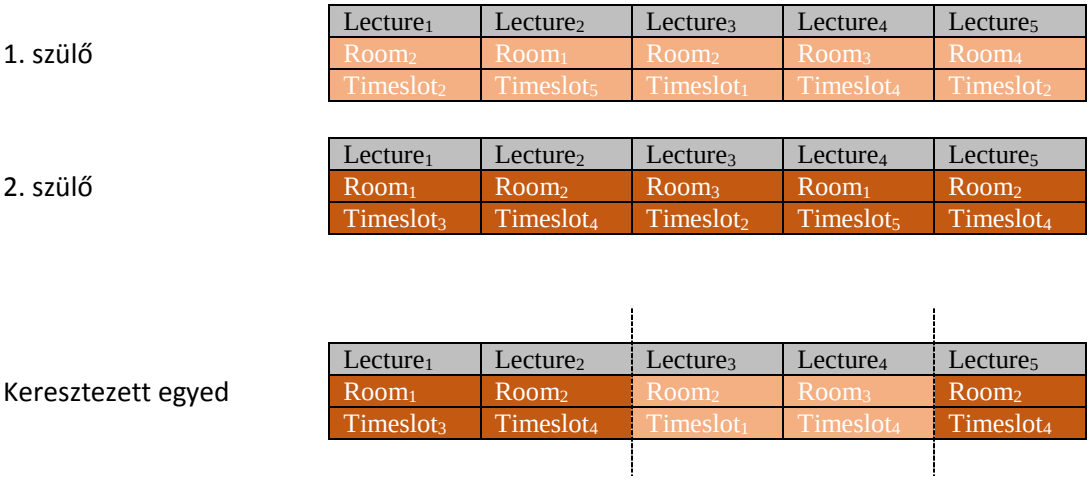
Az egy pont körüli keresztezés lényege, hogy kisorsolunk egy véletlenszerűen választott pontot. A pont előtti géneket az egyik, az utána lévőeket pedig a másik szülőtől vesszük, ahogy az 5. ábrán is látható.



5. ábra: Egy pont körüli keresztezés

5.4.2 Két pont közti keresztezés

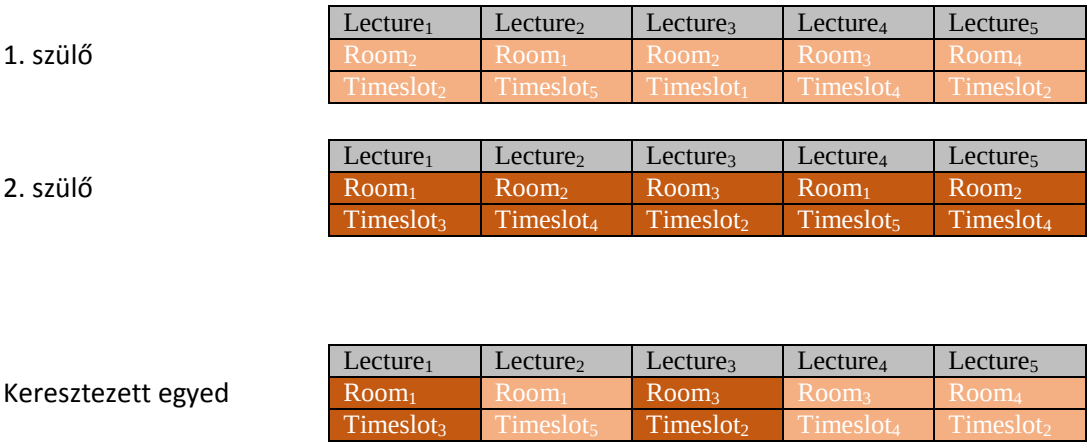
A két pont közti keresztezés hasonlít az előzőre, azzal a különbséggel, hogy két véletlenszerűen választott pont közt lesznek az egyik szülőből választva a gének, a többi pedig a másikból. Ennek eredményét szemlélteti a 6. ábra.



6. ábra: Két pont közti keresztezés

5.4.3 Uniform keresztezés

Az uniform keresztezésnél minden egyes gén véletlenszerűen kerül kiválasztásra valamelyik szülőtől. A 7. ábrán látható, hogy a gének rendre az egyik vagy másik szülőtől kerülnek kiválasztásra.



7. ábra: Uniform keresztezés

5.4.4 Egyéni keresztezés

Bevezetek egy egyéni módot is a keresztezésre. Az uniform keresztezést kibővítem úgy, hogy nem a teljes gént másolom le valamelyik szülőtől, hanem a gén mindkét része, a terem és az időpont véletlenszerűen választott őstől származik, de nem felétlen ugyanattól. Ennek működése a 8. ábrán látható.

1. szülő

$Lecture_1$	$Lecture_2$	$Lecture_3$	$Lecture_4$	$Lecture_5$
$Room_2$	$Room_1$	$Room_2$	$Room_3$	$Room_4$
$Timeslot_2$	$Timeslot_5$	$Timeslot_1$	$Timeslot_4$	$Timeslot_2$

2. szülő

$Lecture_1$	$Lecture_2$	$Lecture_3$	$Lecture_4$	$Lecture_5$
$Room_1$	$Room_2$	$Room_3$	$Room_1$	$Room_2$
$Timeslot_3$	$Timeslot_4$	$Timeslot_2$	$Timeslot_5$	$Timeslot_4$

Keresztezett egyed

$Lecture_1$	$Lecture_2$	$Lecture_3$	$Lecture_4$	$Lecture_5$
$Room_1$	$Room_2$	$Room_2$	$Room_3$	$Room_2$
$Timeslot_3$	$Timeslot_5$	$Timeslot_2$	$Timeslot_4$	$Timeslot_2$

8. ábra: Egyéni keresztezés

5.5 Mutáció

A mutáció során egy kromoszómának bármely génjét véletlenszerű eséllyel változtatom meg. A változás során vagy a termet, vagy az időpontot változtatom meg egymástól függetlenül, így akár a kettő egyszerre változhat. A megoldásom során két fajta mutációs operátort próbáltam ki:

- Gének véletlenszerűsítése
- Gének tulajdonságainak cseréje

Az operátorokon belül megpróbálkozok célirányosabb mutációkat is létrehozni. Ennek az a lényege, hogy heurisztikákat alkalmazva azokat a géneket fogom előbb módosítani, melyek a költségben szerepelnek, így nagyobb eséllyel fogom a költséget csökkenteni. Az alábbi heurisztikákat próbálom ki:

- Véletlenszerű mutáció
- Kemény kényszert először
- Célirányos mutáció
- Célirányos mutáció, kemény kényszert először

A mutációs operátorok mindegyikét leteszteltem, mely során a mutáció-keresztezés párosok egymáshoz képesti viselkedését figyeltem meg. A tesztelés során az összes mutációt lefuttattam mind a négy keresztezéssel összesen ötször, majd ezek átlagos

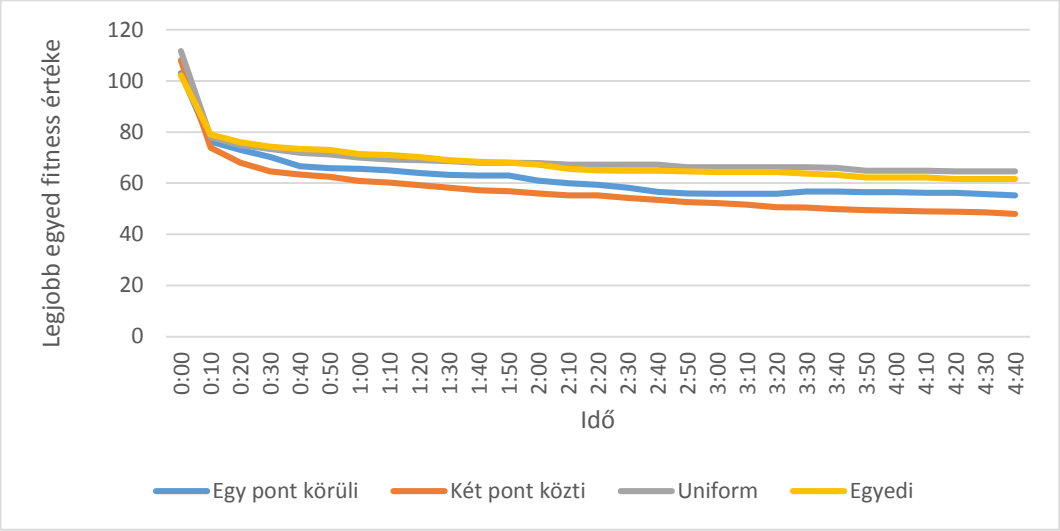
viselkedését figyeltem meg. Az összes diagramon vízszintes tengelyen az eltelt idő, függőleges tengelyen a populáció legjobb egyedének a fitness értéke látható.

5.5.1 Gének véletlenszerűsítése

A gének véletlenszerűsítése során az összes szóba jöhető érték közül választok ki egyet a génnek.

5.5.1.1 Véletlenszerű mutáció

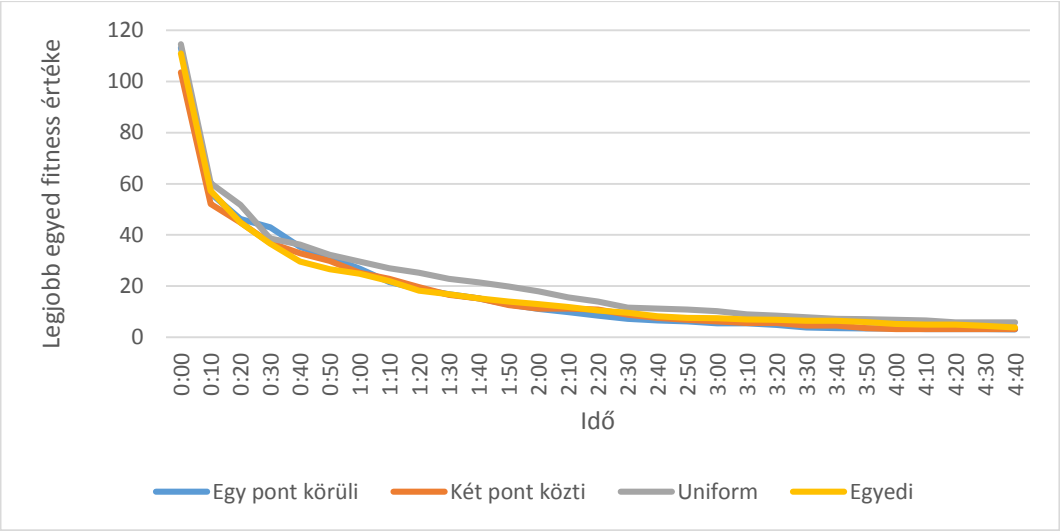
A legegyszerűbb mutáció, mely során minden egyes gén adott valószínűséggel termet cserél és ugyanekkora valószínűséggel időpontot is. Mind a négy keresztezéssel való tesztelés eredményét a 9. ábra mutatja. Egyszerűsége miatt ezt használom referenciának.



9. ábra: Véletlenszerű mutáció

5.5.1.2 Kemény kényszert először

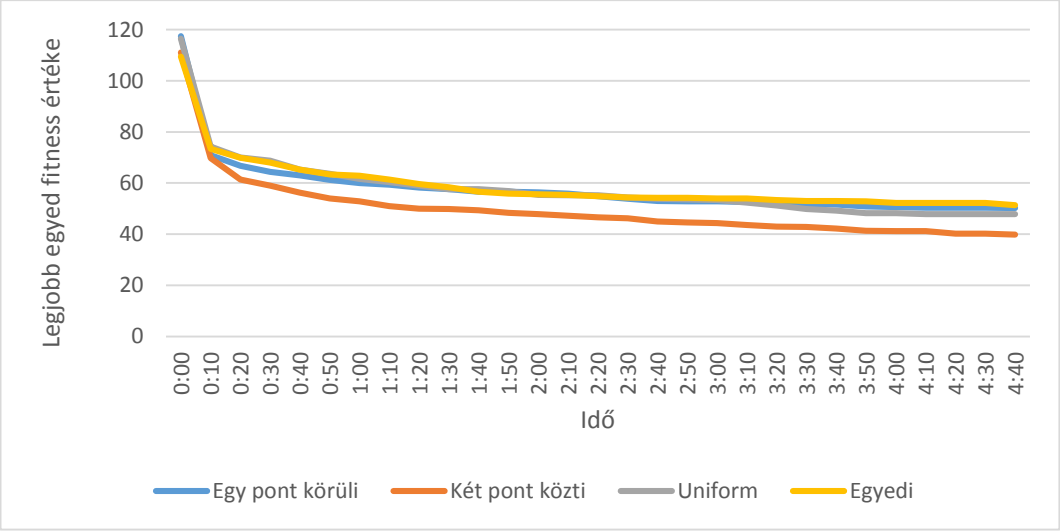
A mutáció során akkor változtatunk meg adott valószínűséggel egy gént, ha az kemény kényszert sért. Ha már nincs ilyen, akkor a laza kényszert sértőket változtatjuk meg ugyanekkora valószínűséggel. A megoldás eredményét a 10. ábra mutatja. Látható, hogy a megoldásom jelentősen javult az előző megoldáshoz képest.



10. ábra: Véletlen mutáció, kemény kényszert először

5.5.1.3 Célirányos mutáció

A mutáció során a génen belül akkor változtatunk meg adott valószínűséggel egy termet, ha az sért egy teremmel kapcsolatos kényszert. Időpontot pedig akkor, ha más fajta kényszert sért. Ahogy 11. ábra is mutatja, ez jobb, mint a legelső, referencia megoldás.

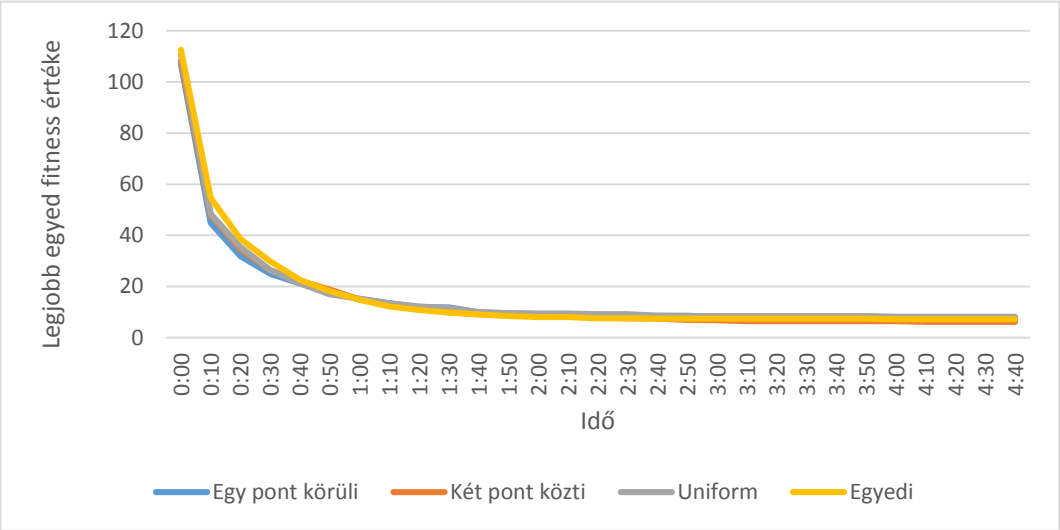


11. ábra: Véletlenszerű mutáció, célirányosan

5.5.1.4 Célirányos mutáció, kemény kényszert először

Az előző két mutáció vegyítése. Először a kemény kényszert sértőket, majd ha nincs ilyen, a laza kényszert sértőket változtatjuk meg úgy, hogy figyelembe vesszük, hogy a terem vagy az időpont sért-e kényszert. Az eredménye a 12. ábra látható. A grafikon alapvetően meredekebben indul, azaz a kezdeti javítása jobb, azonban a végén

rosszabb eredményeket produkál, mint ugyanez kemény kényszert először megszüntető mutáció.



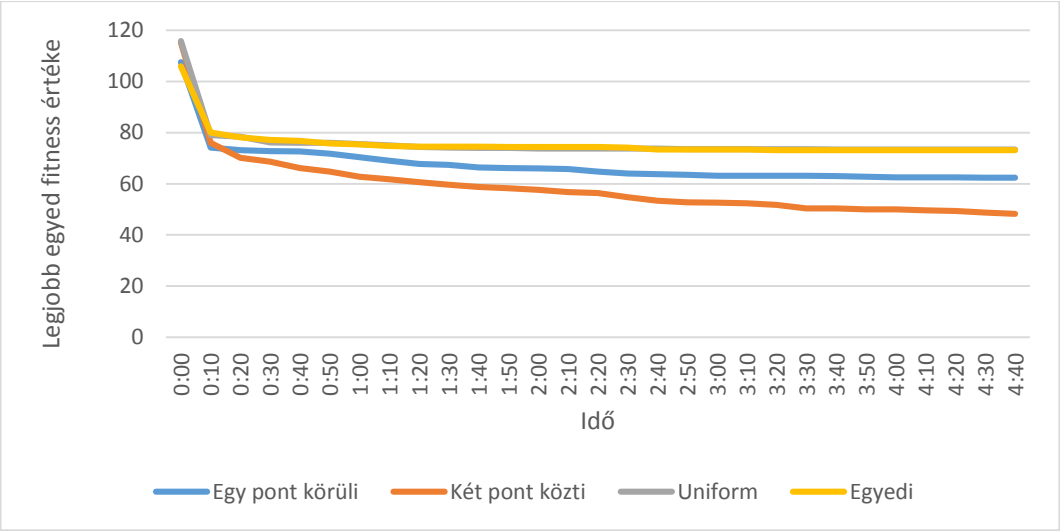
12. ábra: Véletlenszerű mutáció, célirányosan, kemény kényszert először

5.5.2 Gének tulajdonságainak cseréje

A következő mutációk esetén a géneket nem egy véletlenszerű értékre változtatjuk meg, hanem egy másik gént választunk és a kettő tulajdonságait cseréljük meg. A csere során mindig véletlenszerűen választunk egy másik gént.

5.5.2.1 Véletlenszerű mutáció

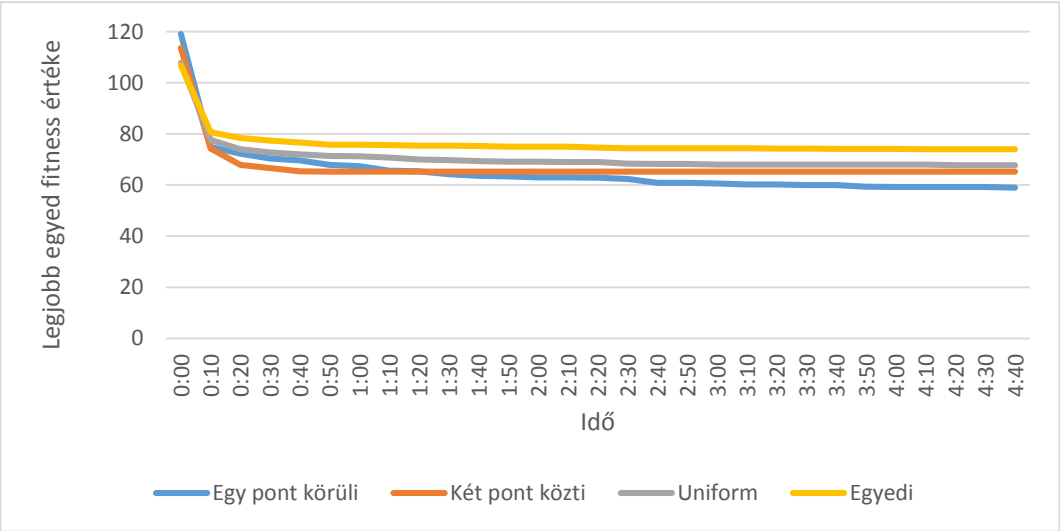
Minden egyes gén esetén adott valószínűséggel választunk egy másik gént, amivel kicserélhetjük egy-egy tulajdonságát. A teszt eredményei a 13. ábra láthatóak, érdekessége, hogy az uniform és egyedi keresztezések nagyon kis különbséget produkáltak.



13. ábra: Csere alapú mutáció, véletlenszerűen

5.5.2.2 Kemény kényszert először

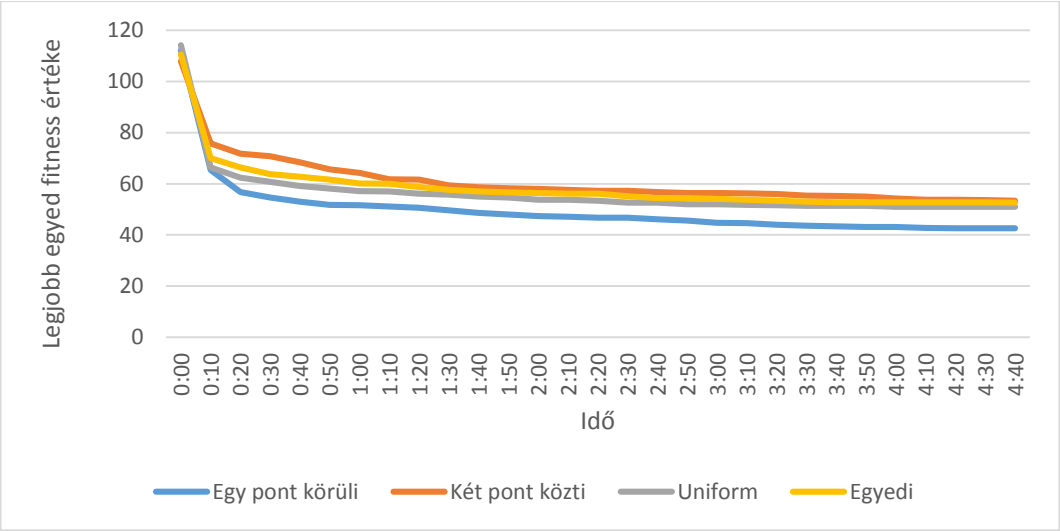
A mutáció során adott vaslószínűséggel akkor cserélünk egy gént, ha az kemény kényszert sért. Ha már nincs ilyen, akkor a laza kényszert sértőket cseréljük. Látható , hogy ennél a mutációnál nem hozott akkora javulást a kemény kényszert először módosító változat. Az eredményt a 14. ábra mutatja.



14. ábra: Csere alapú mutáció, kemény kényszert először

5.5.2.3 Célirányos mutáció

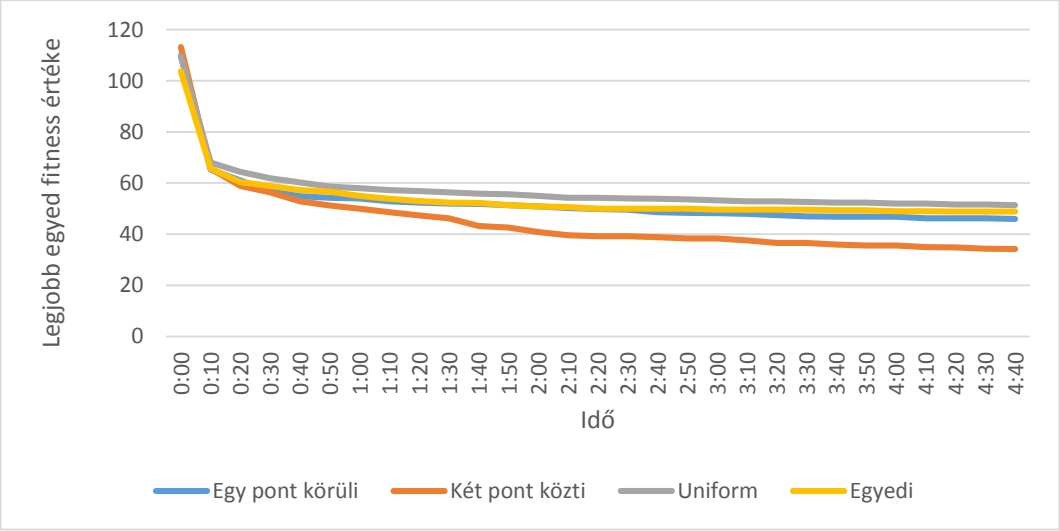
A mutáció során a génen belül akkor cserélünk adott valószínűséggel egy termet, ha az sért egy teremmel kapcsolatos kényszert. Időpontot pedig akkor, ha más fajta kényszert sért. A tesztelés eredményeit a 15. ábra mutatja.



15. ábra: Csere alapú mutáció, célirányosan

5.5.2.4 Célirányos mutáció, kemény kényszert először

Az előző két mutáció vegyítése. Először a kemény kényszert sértőket, majd ha nincs ilyen, a laza kényszert sértőket cseréli meg egy másik génnel úgy, hogy figyelembe veszi, hogy a terem vagy az időpont sért-e kényszert. A 16. ábra által mutatott eredmény szerint ez a mutáció a legjobb a csere alapú megközelítést alkalmazók közül.

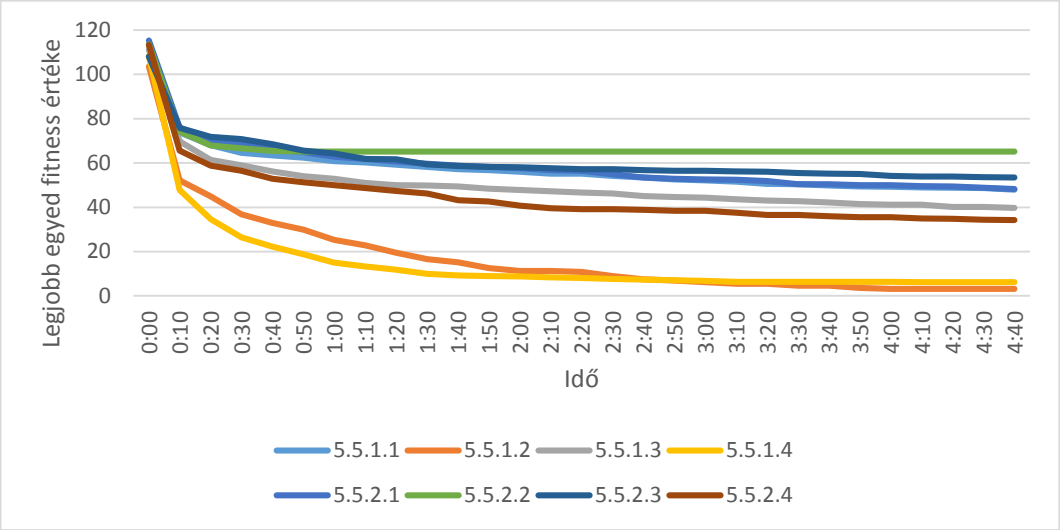


16. ábra: Csere alapú mutáció, célirányosan, kemény kényszert először

5.5.3 Keresztezés és mutációs operátorok tesztelése

A mutációk és keresztezések vizsgálata során látható, hogy az egyes mutációknál a két pont közti keresztezés adta a legjobb eredményeket. A mutációk egymáshoz képesti viselkedését ezzel a keresztezéssel vizsgálom, az eredmény pedig a 17. ábra mutatja. Mivel a feladat a legjobb egyed megtalálása, így nem csupán a mutációk viselkedését kell

figyelembe venni, hanem a legjobb egyed megtalálása a cél. Emiatt az egyes esetekben megvizsgáltam, hogy mi az öt futás közül a legjobb egyed fitness értéke. Ezt az 1. táblázatban foglaltam össze.



17. ábra: A mutációk összehasonlítása

Mutáció fajtája		Egy pont körüli	Két pont közötti	Uniform	Egyedi	Átlagos érték
Gének véletlenszerűsítése	Véletlenszerű	49,1919	42,1636	63,2313	59,1854	53,44305
	Keményet először	0,2149	0,2180	3,2200	1,2195	1,2181
	Célirányos	45,1994	36,1867	41,1910	48,1836	42,6902
	Célirányos, keményet először	3,2380	4,2073	6,1717	5,2320	4,7123
Gének felcserélése	Véletlenszerű	60,2126	47,2147	70,1866	70,1605	61,9436
	Keményet először	55,1684	60,1782	65,1588	69,1793	62,4212
	Célirányos	37,1890	51,2223	43,1701	46,2149	44,4491
	Célirányos, keményet először	42,1831	28,1417	45,2143	45,1584	40,1744
Átlagos érték		40,5735	31,9374	43,1995	42,6861	

1. táblázat: Mutációk és keresztezések legjobb eredményei

A tesztelés értékelése:

A mutációkat összehasonlítva megállapítom, hogy a gének cseréjét alkalmazó mutációk rosszabbul teljesítettek, mint a véletlent alkalmazók. A véletlenszerűsítésen belül először a kemény kényszert sértő gének változtatása radikális mértékben javította a

megoldásomat. A lefuttatott tesztesetek közül ez a módszer kétszer is előállított olyan órarendet, mely nem sértett kemény kényszert.

Méréseim szerint a keresztezések közül átlagosan a két pont közötti érte el a legjobb eredményeket, legrosszabbul pedig az uniform keresztezés teljesített.

Ennek következtében a további teszteket a legjobban teljesítő operátorok segítségével fogom végrehajtani: a két pont közötti keresztezéssel és az először a kemény kényszereket véletlenszerűsítő mutációval.

5.6 Szelekció

A szelekciós operátor segítségével kiválasztom azokat az egyedeket, melyeken a keresztezés illetve mutációs operátort alkalmazom. Az alábbi szelekciós operátorokat tesztelem:

- Egyenletes eloszlású
- Elitizmus
- Tournament
- Rulett

5.6.1 Egyenletes eloszlású szelekció

Ez a szelekció az egyedek információjától függetlenül választ ki egyenletesen elemeket. Az elve egyszerű: generálok egy véletlen számot, majd az annak megfelelő indexű egyed lesz a kiválasztott egyed. Ezt a kiválasztást tekintem a referenciának, ehhez képest vizsgálom, hogy a többi szelekciós módszer javításának a mértékét.

5.6.2 Elitizmus

A legegyszerűbb szelekciós operátor az evolúciós „a legrátermettebb túlélése” elvén működik, ami azt jelenti, hogy a legjobb egyed módosítás nélkül megmarad. De Jong [15] megoldása az volt, hogy az új generáció legrosszabb egyedét kicseréli a régi generáció legjobb egyedére, így annak tulajdonságai nem vesznek el a keresztezés és mutáció során. Ez tulajdonképpen nem olyan szelekciós operátor, mint a többi, mivel ennek a célja nem egy szülő egyed kiválasztása.

5.6.3 Tournament szelekció

Ez a szelekció egy bajnoksághoz hasonlít, ahol a kiválasztásra a legnagyobb fitness értékkel rendelkező egyednek lesz a legnagyobb esélye. Választok egy számot, mely a bajnokság mérete lesz, majd feltöltöm az aktuális generációból véletlenszerűen kiválasztott elemekkel. Ennek a bajnokságnak a legerősebb egyede lesz a szelekció eredménye.

5.6.4 Rulett szelekció

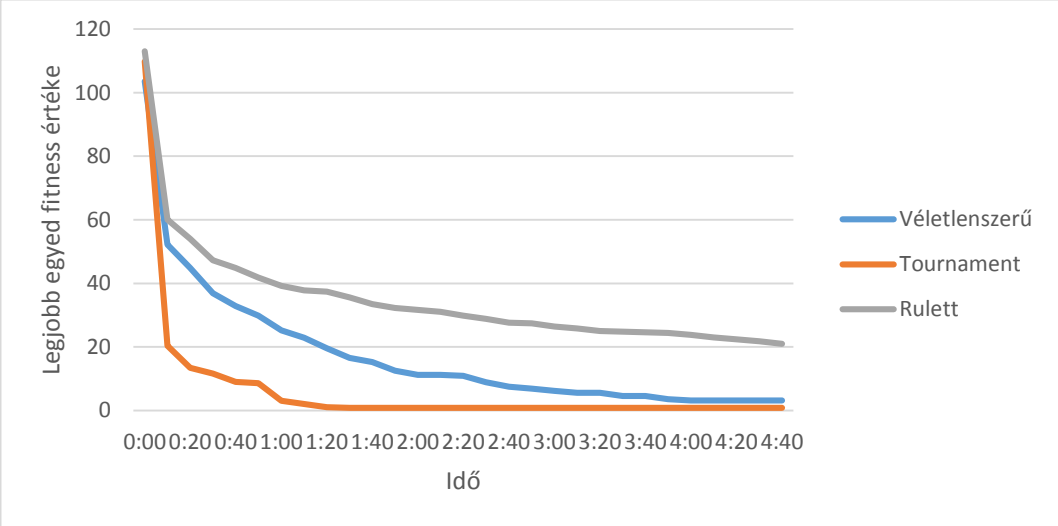
Ebben az esetben minden egyes egyednek egymáshoz viszonyítva pontosan annyi esélye van a kiválasztásra, mint amennyi egymáshoz viszonyított fitness értékük. Ehhez összeadjuk az összes egyed fitness értékét, majd akkora valószínűséggel választjuk ki az egyedet, amennyi saját fitness értékének és az összegnek a hányada:

$$P(e_i) = \frac{f(e_i)}{\sum f(e_j)}$$

ahol *f* az egyedből költséget számító függvény, e_i az i. egyed.

Ezt a szelekciót úgy implementálom, hogy veszek véletlenszerűen egy egyedet, majd adott valószínűséggel kiválasztom, addig ismételve, amíg nem sikerül egyet kiválasztani. Ez az algoritmus átlagosan O(1) komplexitású [16].

5.6.5 Szelekciós operátorok tesztelése

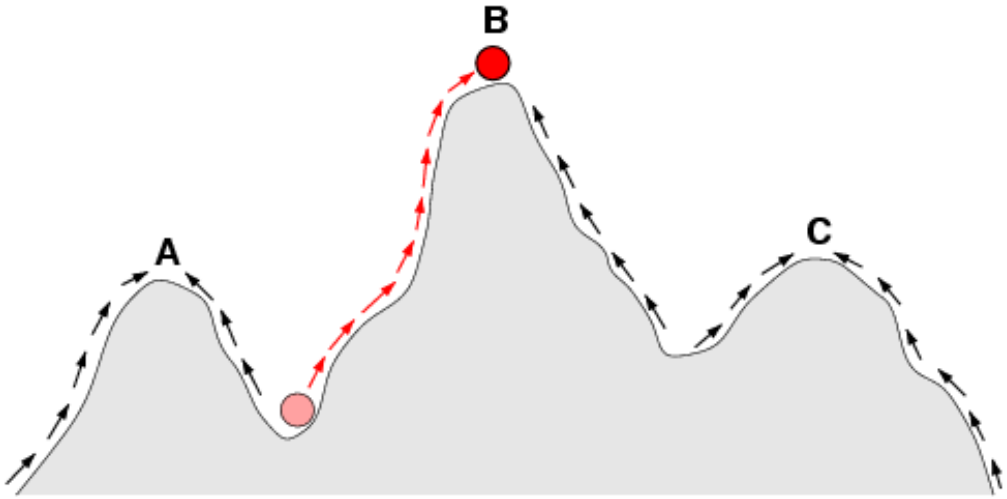


18. ábra: Szelekció operátorok összehasonlítása

Miután a szelekciós operátorokat implementáltam a fenti, legjobb keresztezés és mutáció segítségével teszteltem azok konvergenciáját. A tesztelés eredményei a 18. ábra mutatja. Itt is ötször futtattam le a tesztesetek, ahol a keresztezés az 5.4.2 pontban ismertetett két pont közti keresztezés, a mutáció pedig az 5.5.1.1 pontban ismertetett, véletlenre épülő mutáció volt. A vízszintes tengelyen itt is az idő, a függőlegesen pedig a legjobb egyed fitness értéke látható,

5.7 Lokális optimum elkerülése

Genetikus algoritmusoknál gyakori eset, hogy a konvergálás nem egy globális optimumhoz történik, hanem csak egy lokális optimumhoz. Ezt úgy kell elképzelni, mint amikor a hegymászó keresés a legmagasabb csúcsot keresi. A 19. ábrán látható B pont globális optimum, míg A és C lokálisak. A keresés során mindig arra haladunk tovább, amerre emelkedik a saját magasságunk. Amikor több irány is lehetséges döntést kell hoznunk, hogy merre megyünk tovább. Ideális esetben B pontba jutunk el, de egy döntéssorozat eredményeképpen az A pontba is eljuthatunk. Azonban amikor A pontba eljutunk, nem tudunk ezzel az algoritmussal továbbjutni.



19. ábra: Legmagasabb csúcsok keresése [17]

Ezt nevezzük lokális optimumban ragadásnak, melyet különböző problémák esetén különböző módon próbálnak kivédeni. Genetikus algoritmus esetén ez akkor történik meg könnyen, ha a megoldásunk kezd a legjobb megoldás közelébe jutni. Ebben az esetben a megoldásunk már elég jó ahhoz, hogy bármerre ellépve rosszabb megoldást kapjunk, mely nagy valószínűséggel a szelekció után elveszik, de még nem éri el az

optimálist. Az algoritmus javulása során egyre többször került az enyém is ilyen helyzetbe. Annak érdekében, hogy ezt elkerüljem, az alábbi módszereket próbáltam ki:

- Visszalépés
- Új egyedek keresztezésnél
- Erősödő mutáció

5.7.1 Visszalépés

A visszalépés lényege, hogy amikor a megoldásunk már nem javul, akkor az adott legjobb megoldást félretéve visszalépünk egy korábbi megoldáshoz, és azt a keresést próbáljuk meg más irányba elvinni. Ezt úgy implementáltam, hogy amikor javul a populáció legjobb egyede, azt elmentem egy listába melyet akkor veszek igénybe, amikor ki akarok kerülni a lokális optimumból. Ekkor az új generációt a félretett egyedekből hozom létre.

5.7.2 Új egyedek

A genetikus algoritmus akkor képes hatékonyan keresni, ha az egyedek varianciája nagy. Lokális optimumban ragadás esetén a legtöbb egyed egyforma lesz, és ezek keresztezésekor ugyanaz az egyed fog létrejönni, így nincs olyan eset, hogy a keresztezés javítana a populáción.

Ebben a megoldásban úgy döntök, hogy a két egyed egyezősége esetén az egyik helyett új egyedet fogok létrehozni és azzal keresztezem a kromoszómát. Mivel a teljes vizsgálat költséges és sokszor felesleges is, így úgy döntök, hogy a két kromoszóma egyező fitness értéke esetén akkora valószínűséggel fogom ezt a módszert alkalmazni, amennyi generáció óta nem változott az eredmény.

5.7.3 Erősödő mutáció

A lokális optimumban ragadásnak az az okozója, hogy a kis változások már egyáltalán nem javítanak megoldáson, és elvetjük őket, mielőtt esetleg azokból kiindulva tovább kereshetnénk abba az irányba. Annak érdekében, hogy elegendő mértékű legyen a változtatás mértéke, erősebb mutációra van szükség.

Ha bizonyos kényszereket sért, akkor az alábbi mutációkkal próbálom a termeket cserélni:

- Teremütközés: az adott időpontban megpróbálom az összes génhez különböző termet rendelni, ha ez nem lehetséges, akkor abból az időpontból elrakom máshova a géneket, amíg nem lesz megoldható.
- Teremkapacitás: az adott időpontban található gének termeit minimalizálom
- Teremhasználat: az adott tárgyhoz tartozó összes gén ugyanazt a véletlenszerű termet kapja meg

Az időpontokat pedig az alábbi módon módosítom:

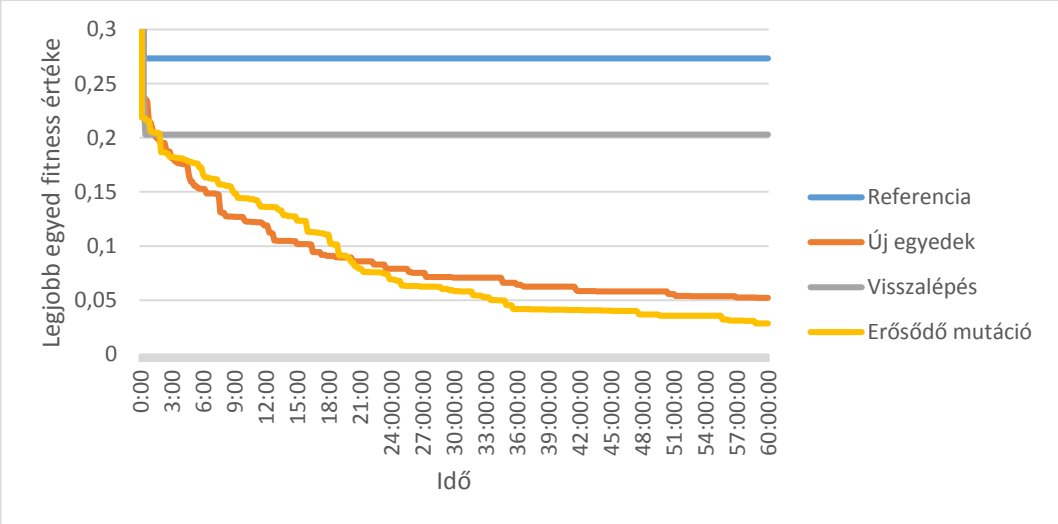
- Ütközés: az ütköző órák valamelyikét vagy mindegyikét más időpontra rakom
- Rendelkezésre állás: a gén időpontját egy véletlenszerűvel kicserélem
- Elszigetelt órák: egy másik elszigetelt óra mellé rakom

Az ütközést nem okozó géneket pedig nagyobb valószínűséggel módosítom.

Ezt a mutációt végrehajtom 50 egyeden, akiket a potenciális szülőkromoszómákhoz adok.

5.7.4 Lokális optimum elkerülésének tesztelése

Annak érdekében, hogy a lokális optimumban ragadást megfigyelhessem, elég hosszúnak kell lennie a vizsgált időintervallumnak. Ennek érdekében én a referenciamódszert, valamint a fenti változtatások utáni algoritmust 1 órán keresztül figyeltem és azt vizsgáltam, hogy lokális optimumban ragadnak-e.



20. ábra: Lokális optimum elkerülése

Az 20. ábrán szemléltetem egy órán keresztül történő futás eredményét. Az eddigiekhez hasonlóan a vízszintes tengelyen az idő, a függőlegesen a populáció legjobb egyedének a fitness értéke látható. A megoldások mindegyike elérte a 0 kemény költséget, de kiderült, hogy az alap algoritmus lokális optimumban ragad, ugyanúgy, ahogy a visszalépés esetén is. Az új egyedekkel való keresztezés, illetve az erős mutációk hatékony eszköznek bizonyultak az optimumban ragadás leküzdésére. A teszt eredményének oka az egyedek sokszínűségéből vezethető le. Mivel az eredeti keresztezés nem hoz be új egyedeket, az nem fogja a lokális optimumot megszűntetni. A választott mutáció a kényszert sértő géneket próbálja meg módosítani, de olyan esetben, ahol több gén, akár igen komplex megváltoztatása szükséges, ott ez nem javít. Mindkét működő esetben olyan gének kerülhetnek a szülők közé, melyek sosem szerepeltek még, így segítségükkel olyan megoldások is előállhatnak, melyeket amúgy nem vennénk számításba.

5.8 Javasolt algoritmus

5.8.1 Algoritmus Részletezése

Az eddigi teszteseteteket figyelembe véve az alábbi algoritmust ajánlom:

0. Véletlenszerűen létrehozok 100 egyedet
1. Létrehozok egy új generációt, melybe belehelyezem a populáció legjobb egyedét
   1.1. Tournament szelekcióval kiválasztok két egyedet, melyeket keresztezni fogok
      1.1.1. Létrehozok két pont közti keresztezéssel két egyedet, alkalmazva az 5.7.2 pontban ismertetett új egyed hozzáadását
      1.1.2. Kiszámolom az egyedek fitness értékét
      1.1.3. Az egyedeket az új generációba rakom
   1.2. Tournament szelekcióval kiválasztok egy egyedet, amin az alábbi mutációt fogom alkalmazni
      1.2.1. Alkalmazom a véletlenszerű, először kemény kényszert változtató mutációt
      1.2.2. Kis eséllyel alkalmazom az 5.7.3-ban alkalmazott mutációk egyikét
      1.2.3. Kiszámolom az új egyed fitness értékét
      1.2.4. Az új egyedet az új generációba rakom
   1.3. Ha még nincs annyi egyed az új generációban, mint a régiben, akkor az 1-es ponttól folytatom
2. Az új generáció lesz az aktuális populáció
3. Ha 100 generáció óta nem javult az eredmény, az 5.7.3 pontban ismertetett teljes mutációt alkalmazom a lokális optimumban ragadás ellen
4. Ha nem értem el a limitidőt, akkor az 1-es ponttal folytatom
5. Megkeresem a legjobb kromoszómát, és előállítom a kész órarendet belőle

5.8.2 Kapott eredmény

Ötször futtattam az algoritmust az első tesztesetre, melyek eredményei az 2. táblázatban láthatóak.

Futás száma	Kemény költség	Laza költség
1. futás	0	799
2. futás	0	876
3. futás	0	424
4. futás	0	453
5. futás	0	858

2. táblázat: Első teszteset megoldása

Láthatóan az algoritmus az összes teszteseten tudott javítani a véletlen órarendhez képest. Ezen felül képes volt mindegyik futási esetben megvalósítható órarendet adni, azaz egy kemény kényszert sem értő megoldásokat találni.

6 Algoritmusok értékelése

6.1 Tesztesetek

A teszteseteket a 3. táblázat foglalja össze, melyben megtalálható, hogy mennyi erőforrást (órát, termet, időpontot) kellett beosztani, és mennyi kényszer okozhatott ütközést az órák között. Mivel az összes teszteset azonos formátummal rendelkezik, így az ajánlott algoritmus képes megoldásokat találni azokra is. A tesztelés a 4 pontban ismertetett módszerrel zajlik. Az elsődleges cél minden esetben egy megvalósítható megoldás elkészítése volt.

Teszteset neve	Órák száma	Termek száma	Időpontok száma	Tanárok száma	Tantervek száma
comp01	126	6	30	24	14
comp02	283	16	25	71	70
comp03	251	16	25	61	68
comp04	286	18	25	70	57
comp05	152	9	36	47	139
comp06	361	18	25	87	70
comp07	434	20	25	99	77
comp08	324	18	25	76	61
comp09	279	18	25	68	75
comp10	370	18	25	88	67
comp11	162	5	45	24	13
comp12	218	11	36	74	150
comp13	308	19	25	77	66
comp14	275	17	25	68	60
comp15	251	16	25	61	68
comp16	366	20	25	89	71
comp17	339	17	25	80	70
comp18	138	9	36	47	52
comp19	277	16	25	66	66
comp20	390	19	25	95	78
comp21	327	18	25	76	78

3. táblázat: Tesztesetek tulajdonságai

6.2 Tesztek eredménye

A teszteket az előzőekhez hasonlóan ötször futtattam egymás után 286 másodpercig, majd az öt teszteset során kapott legjobb eredményeket vettem figyelembe. Mindegyik tesztesethez lefuttattam az 5.1-es pontban leírt javított véletlenszerű algoritmust, a

genetikus algoritmus hasonlóan, 286 másodperces időkorláttal. Ezután a legjobb elkészült megoldást mentettem el és ehhez képest vizsgáltam a genetikus algoritmus javítását.

A 4. táblázatban láthatóak a teszt eredményei. Az algoritmus mind a 21 teszteset képes volt nagymértékben javítani a referencia-megoldáson, és 8 esetben olyan órarendet készített, mely megvalósítható volt.

Teszteset neve	Véltetlen kemény költség	Véltetlen laza költség	Genetikus kemény költség	Genetikus laza költség
comp01	67	2357	0	424
comp02	127	7562	5	3111
comp03	117	4701	3	5201
comp04	122	5641	0	1773
comp05	73	9166	5	5101
comp06	303	5397	5	2682
comp07	189	6806	8	2348
comp08	228	6941	0	1889
comp09	154	4916	0	3028
comp10	205	5825	4	2477
comp11	59	2701	0	437
comp12	139	3674	4	3136
comp13	129	7829	0	1852
comp14	150	4214	0	1562
comp15	121	5387	3	2843
comp16	177	4331	3	2015
comp17	184	5487	2	2310
comp18	84	2087	0	424
comp19	137	6354	2	2978
comp20	197	8632	4	7204
comp21	168	7048	5	3037

4. táblázat: A tesztesetek eredményei

6.3 Fejlesztési lehetőségek

A tesztesetekből látszik, hogy nem sikerült az adott időn belül megfelelő megoldást adni, az időlimit növelésével azonban minden esetben sikeresen előállítottam olyan megoldásokat, melyek nem sértettek kemény kényszert. Ennek következtében a legfontosabb fejlesztés a megoldás gyorsítását célozza.

Párhuzamosítás: a genetikus algoritmus egy nagyon jól párhuzamosítható feladat. A tesztelés során előfordult öt futás sokszor nagyon eltérő lehet, így a legegyszerűbb párhuzamosítás ezen populációk egyszerre történő futtatása. A populációk közti átjárás is biztosíthatja a gyorsítást, amennyiben egy jobb egyed kerül egy populációba. Párhuzamosítható a populáció futtatásán belül is az új egyedek létrehozása

is, azaz a genetikus operátorok és a fitness kiértékelő folyamat is, ugyanis egy-egy generáción belül ezek egymástól teljesen függetlenül hajthatóak végre.

Számítási felhő: manapság egyre több helyen lehet bérelni számítási felhőt rövid időre, ahol a program gyors, és erős hardverrel rendelkező számítógépeken futhat. Mivel az órarendtervezés egy ritkán felmerülő feladat, így ezekre az alkalmakra ki lehet bérelni ilyen felhőket.

GPGPU (GPU általános programozása): a videokártyák teljesítményének rohamos fejlődésével egyre több algoritmust implementálnak a grafikus kártyák sajátosságait kihasználó algoritmusokat, mint például a jelszóval védett tömörített állományok feltörését. A genetikus algoritmus, a párhuzamosíthatóság miatt, alkalmas lehet GPU-n történő futtatásra is megfelelő implementáció esetén.

Vegyes módszer: az algoritmus futása során a megoldások nagyon gyorsan javultak, majd az idő előrehaladtával, az optimumhoz közeledve, egyre kisebb volt a javítás mértéke. Ennek érdekében egy járható út, hogy az elején egy nagyobb javítást elvégezve egy-egy potenciálisan jobb megoldásból kiindulva más módszer igénybevételével próbálunk meg keresést indítani. Ilyen módszerek lehetnek más, már említett, metaheurisztikát alkalmazó keresések, esetleg a lokális vagy a szomszédossági keresések is.

7 Összefoglalás

A dolgozat célja a genetikus algoritmus egyes megvalósításainak összehasonlítása, és ennek segítségével egy algoritmus készítése volt, mellyel megoldhatóvá vált a verseny során kiírt órarend-tervezési probléma.

Először naiv megoldásokkal véletlenszerűen próbálkoztam előállítani órarendeket, de ezzel a módszerrel nem sikerült megvalósítható órarendeket előállítanom. A következő lépésben egyszerű módosításokkal képes voltam javítani az eddigi eredményeken.

Ezután megvizsgáltam, hogy genetikus algoritmus segítségével hogyan lehet órarendeket előállítani. Implementáltam egy keretrendszert, mely képes volt a problémákat beolvasni, azokat feldogozni, majd egy megoldást megfelelő formátumban előállítani. Az előzőleg készített órarendeket referenciaként számon tartva igyekeztem jobb megoldásokat készíteni. Ennek érdekében az általánosan igénybevett genetikus operátorokat, a keresztezést, a mutációt és a szelekciót, különböző módokon megvalósítottam, és ezek viselkedését figyeltem meg. A genetikus algoritmus tesztelése során felismertem, hogy a jó megoldásokhoz vezető úton lokális optimumban ragadhatunk, mely ellen különböző módszereket ajánlottam.

Végül egy olyan algoritmust ajánlottam, mely az általam vizsgált tesztesetre jó megoldásokat képes adni a megadott időhatáron belül, és ezen felül képes volt javítani az összes vizsgált tesztesetben az órarendeken. Ezek a megoldások, mivel elkerülik a lokális optimumot, hosszabb futási idő során képesek eljutni az optimumba, de ennek ideje nem determinisztikus. Mivel az órarendtervezés egy ritkán elvégzendő feladat, például egyetemek esetében félévente elég egyszer lefuttatni az algoritmust, így ez a nem determinisztikusság nem okoz problémát.

8 Irodalomjegyzék

[1] S. Even, A. Itai, and A. Shamir „On the complexity of time table and multi-commodity flow problems” *16th Annual Symposium on Foundations of computer science*, pp. 184-193, October. 1975.

[2] D. de Werra, „An introduction to timetabling” *European Journal of Operational Research,* pp. 151–162, 02. 1985.

[3] D. de Werra and A. Hertz, „The tabu search metaheuristic: How we used it” *Annals of Mathematics and Artificial Intelligence,* pp. 111-121, September 1990.

[4] Andrea Roll and Christian Blum, „Metaheuristics in combinatorial optimization: Overview and conceptual comparison,” *ACM Computing Surveys (CSUR),* pp. 268-308, September 2003.

[5] Marco Dorigo, Vittorio Maniezzo and Alberto Colorni, „A genetic algorithm to solve the timetable problem,” *Computational Optimization and Applications,* 1993.

[6] K. A. Dowsland and J. M. Thompson, „Ant colony optimisation for the examination scheduling problem,” *Journal of the Operational Research Society,* pp. 426-438, 2006.

[7] Shu-Chuan Chu, Yi-Tin Chen and Jiun-Huei Ho, „Timetable Scheduling Using Particle Swarm Optimization,” *International Conferece on Innovative Computing, Information and Control,* Beijing, pp. 324-327., 2006

[8] A. Hertz, „Tabu search for large scale timetabling problems,” *European Journal of Operational Research,* pp. 39-57, 1991.

[9] E.K. Burke, G. Kendall and E. Soubeiga, „A Tabu-Search Hyperheuristic for Timetabling and Rostering,” *Journal of Heuristics,* pp. 451-470, December 2003.

[10] Jonathan M. Thompson and Kathryn A. Dowsland, „A robust simulated annealing based examination timetabling system,” *Computers & Operations Research,* p. 637–648., July 1998.

[11] J. Sheung; A. Fan and A. Tang, „Time tabling using genetic algorithm and simulated annealing," *Conference on. Computers, Communications, Control and Power Engineering*, Beijing, 1993.

[12] John. H. Holland, „Adaptation in Natural and Artificial Systems", London: The MIT Press, 1992.

[13] „Curriculum Based Course Timetabling," [Online]. Available: http://www.cs.qub.ac.uk/itc2007/curriculmcourse/course_curriculm_index.htm.

[14] „Datasets," [Online]. Available: http://www.cs.qub.ac.uk/itc2007/Login/SecretPage.php.

[15] De Jong and Kenneth Alan, *Analysis of the behavior of a class of genetic adaptive systems,* University of Michigan Ann Arbor, 1975.

[16] Adam Lipowskia and Dorota Lipowskab, „Roulette-wheel selection via stochastic acceptance," *Physica A: Statistical Mechanics and its Applications*, p. 2193–2196., 2012.

[17] „Wikipedia, Fitness landscape," [Online]. Available: http://en.wikipedia.org/wiki/Fitness_landscape.

A Függelék: Bemeneti formátum

A bemeneti fájl mezőinek formái:

- Tárgyak: <TárgyID> <Tanár> <Órák száma> <Min napok> <Diákok száma>
- Termek: <TeremID> <Kapacitás>
- Tanterv: <TantervID> <Tárgyak száma> <$Tárgy_1$> ... <$Tárgy_n$>
- Rendelkezésére állás: <TárgyID> <Nap> <Periódus>

Példa bemenet az első tesztesetre:

```
Name: Fis0506-1
Courses: 30
Rooms: 6
Days: 5
Periods_per_day: 6
Curricula: 14
Constraints: 53

COURSES:
c0001 t000 6 4 130
c0002 t001 6 4 75
…
c0072 t003 6 4 9

ROOMS:
rB 200
rC 100
rE 9
rF 30
rG 20
rS 30

CURRICULA:
q000 4 c0001 c0002 c0004 c0005
q001 4 c0014 c0015 c0016 c0017
…
q013 3 c0062 c0066 c0071

UNAVAILABILITY_CONSTRAINTS:
c0001 4 0
c0001 4 1
…
c0071 4 2

END.
```

B Függelék: Kimeneti formátum

Kimeneti fájl mezőinek

- Óra: <TárgyID> <TeremID> <Nap> <Periódus>

Példa kimenet:

```
c0001 rB 0 4
c0001 rB 1 1
c0001 rB 2 4
c0001 rB 3 1
c0001 rB 3 2
c0001 rB 3 3
c0002 rB 0 5
c0002 rB 1 0
c0002 rB 3 5
c0002 rB 4 1
c0002 rB 4 2
c0002 rB 4 5
c0004 rB 1 2
…
c0072 rG 4 4
```

C Függelék: Ellenőrző program kimenete

A verseny szervezői által biztosított ellenőrző program az alábbi példa kimenetet produkálja:

```
[S(1)] Room rF too small for course c0033 the period 7 (day 1, timeslot 1)
[S(1)] Room rF too small for course c0033 the period 13 (day 2, timeslot 1)
[S(1)] Room rF too small for course c0033 the period 14 (day 2, timeslot 2)
[S(1)] Room rF too small for course c0033 the period 15 (day 2, timeslot 3)
[S(2)] Curriculum q003 has an isolated lecture at period 18 (day 3, timeslot 0)
[S(2)] Curriculum q008 has an isolated lecture at period 12 (day 2, timeslot 0)
[S(2)] Curriculum q012 has an isolated lecture at period 14 (day 2, timeslot 2)
[S(2)] Curriculum q012 has an isolated lecture at period 29 (day 4, timeslot 5)
[S(2)] Curriculum q013 has an isolated lecture at period 13 (day 2, timeslot 1)
[S(1)] Course c0002 uses 2 different rooms
[S(1)] Course c0015 uses 2 different rooms
[S(1)] Course c0016 uses 2 different rooms
[S(1)] Course c0031 uses 2 different rooms
[S(1)] Course c0033 uses 2 different rooms
[S(1)] Course c0058 uses 2 different rooms
[S(1)] Course c0061 uses 2 different rooms
[S(1)] Course c0064 uses 2 different rooms
[S(1)] Course c0065 uses 2 different rooms
[S(1)] Course c0066 uses 2 different rooms
[S(1)] Course c0067 uses 2 different rooms
[S(1)] Course c0068 uses 2 different rooms
[S(1)] Course c0070 uses 2 different rooms
[S(1)] Course c0072 uses 2 different rooms

Violations of Lectures (hard) : 0
Violations of Conflicts (hard) : 0
Violations of Availability (hard) : 0
Violations of RoomOccupation (hard) : 0
Cost of RoomCapacity (soft) : 4
Cost of MinWorkingDays (soft) : 0
Cost of IsolatedLectures (soft) : 10
Cost of RoomStability (soft) : 14

Summary: Total Cost = 28
```