XIV. Műszaki tudományos ülésszak, 2013. Kolozsvár, 165–171. *DOI: 10.33895/mtk-2014.01.16*
http://hdl.handle.net/10598/28089

ÜTEMEZÉSI FELADATOKRA ALKALMAZOTT GENETIKUS ALGORITMUS KERESZTEZŐ OPERÁTORAINAK VIZSGÁLATA

EXAMINATION OF THE CROSSOVER OF GENETIC ALGORITHM APPLIED TO SCHEDULING TASKS

Simon Pál

Miskolci Egyetem, Informatikai intézet, Alkalmazott Informatikai Tanszék 3515 Miskolc, Miskolc-Egyetemváros, pal.simon.8@gmail.com

Abstract

Production scheduling problems are complex order design tasks, whose solution is still a difficult task nowadays. These tasks are usually modelled with the Travelling Salesman Problem. This is an NP-hard problem and can't be solved optimally within a reasonable time. Acceptable but not optimal solution provided by artificial intelligence procedures within a reasonable period of time. Such process is the genetic algorithm. This algorithm has been used for a long time to solve the travelling salesman problem. A key part of the algorithm is the crossover operator, that affecting the efficiency. Over time, many crossing operators were developed. Each of these has similar characteristics such as maintaining the position and the order of the subsequence. In this article I show the disadvantage of these characteristics. Then I present an operator that does not have these characteristics and thus it makes the algorithm more efficient. To support this increased efficiency I will present some test results.

Keywords*:scheduling, genetic algorithm, crossover,travelling salesman problem.*

Összefoglalás

A gyártásütemezési feladatok összetett sorrend előállítási feladatok, amelyek megoldásának előállítása napjainkban is nehéz feladat. Ezeket a feladatokat az utazó ügynök problémával jól lehet modellezni, amely egy NP-nehéz probléma, és nem oldhatók meg optimálisan ésszerű időn belül. Nem optimális, de elfogadható megoldást lehet elérni elfogadható időn belül mesterséges intelligencia módszerekkel. Egy ilyen módszer a genetikus algoritmus. Ezt a módszert már régóta alkalmazzák az utazó ügynök probléma megoldására. Az algoritmus egyik legfontosabb része az egyedek keresztezése, amely nagymértékben befolyásolja a hatékonyságát. Több keresztező operátort is kifejlesztettek napjainkra. Ezek mindegyike hasonló jellemzőkkel rendelkezik, mint például a keresztezni kívánt egyedekben lévő részsorrendek pozícióinak és sorrendjének megőrzése. Ebben a cikkben ezekre a jellemzőkre szeretnénk rávilágítani. Majd egy olyan operátort mutatunk be, amely nem rendelkezik ezekkel a jellemzőkkel, ezáltal hatékonyabbá teszi az algoritmust. Ezt a hatékonyságnövekedést teszteredmények bemutatásával támasztjuk alá.

Kulcsszavak: *ütemezés, genetikus algoritmus, keresztező operátor, utazó ügynök probléma.*

1. Bevezetés

Napjainkban egyre fontosabb az előállított termékek jó minősége és alacsony gyártási költsége. Ezeknek az elvárásoknak való megfelelés miatt nagyon fontos a korszerű technikák alkalmazása a gyártás során, ezáltal növelve a hatékonyságot a gyártási folyamat során felmerülő költségek csökkentésével. A gyártási költségek csökkentésének egyik leghatékonyabb módja a gyártó kapacitás legjobb kihasználása. Ennek elérése érdekében a gyártásütemezés hatékonyságát kell növelni.

A gyártásütemezés során felmerülő összetett ütemezési problémák modellezésére használható az utazó ügynök probléma [1–3]. Ami egy NP-nehéz probléma és optimálisan nem oldható meg valós időn belül. Azonban optimálishoz közeli megoldást lehet elérni heurisztikus vagy mesterséges intelligencia módszerekkel valós időn belül. A gyakorlatban ezek az optimálishoz közeli megoldások is elfogadhatók. Ilyen megoldást eredményező mesterséges intelligencia módszer a genetikus algoritmus.

A genetikus algoritmus az egyedpárok keresztezésével állít elő új egyedeket, amelyek pozitív tulajdonságai javulhatnak. A szelekció révén a jobb tulajdonságokkal rendelkező egyedek maradnak meg, így fokozatosan javul a megoldás és közelít az optimum felé.

Az algoritmus hatékonyságát nagymértékben befolyásolja az alkalmazott keresztező operátor. Napjainkra több keresztező operátort is kifejlesztettek. Ezek különbözőképpen keresztezik az egyedpárokat, de mindegyik hasonló azáltal, hogy vagy csak a sorrendet, vagy csak a pozíciót változtatják [4-6]. Feltételezéseink szerint a sorrendtervezési feladatoknál, mint az utazó ügynök probléma, hatékonyabb módszer is létezik.

A cikk első felében a problémát és a megoldására alkalmazott genetikus algoritmust alkalmazó eljárást mutatjuk be. Majd a leggyakrabban alkalmazott keresztező operátorokat és azok tulajdonságait mutatjuk be, rávilágítva azokra a tulajdonságokra, amelyek jellemzőek ezekre az operátorokra. Végül egy új keresztező operátort ismertetünk, amely a meglévő operátoroktól eltérő jellemzőkkel bír. Teszteredményt mutatunk be, amely alátámasztja az általunk bemutatott operátor hatékonyságát.

2. Az utazó ügynök probléma

Az Utazó ügynök probléma egy sorrendtervezési probléma modell. Ezért jól használható különböző ütemezési problémák modellezésére, mint például a gyártásütemezés. [7]. Adott n város és minden város között lévő távolság. A feladat egy olyan legrövidebb bejárási útvonalat keresni, amely minden várost pontosan egyszer érint, és az út végén visszajutunk a kiindulási pontba [8].

Az utazó ügynök probléma a következőképpen írható le. Adott n város halmaza $N=\{1, \ldots, n\}$. A távolságot i városból j-be jelölje d_{ij}, minden $i \in N$, $j \in N$, $i \neq j$. Az utazó ügynök útvonala legyen a következő $\pi=(\pi(1), \pi(2), \ldots, \pi(n))$ az N halmazra, ahol $\pi(j)$ a j-ediként meglátogatott város. A feladat, hogy találjunk olyan π^* permutációt, ami minimalizálja a teljes út hosszát $D(\pi^*)$, ahol $D(\pi^*)$ a következő:

$$D(\pi) = \sum_{j=1}^{n-1} d_{\pi(j)\pi(j+1)} + d_{\pi(n)\pi(1)} \qquad (1)$$

A feladat nehézségét az adja, hogy a lehetséges bejárások száma n város esetén $(n-1)!/2$. Emiatt a feladat számítási igénye a városok számával exponenciálisan növekszik. Annak érdekében, hogy ésszerű időn belül meg lehessen oldani ezt a problémát heurisztikus vagy közelítő módszereket kell alkalmazni. Ilyen módszereket a mesterséges intelligencia algoritmusok kínálnak, amelyek optimálishoz közeli megoldást

eredményeznek, de furásidejük belátható időn belül marad.

3. Genetikus algoritmus

A genetikus algoritmus egy heurisztikus keresési algoritmus, amely a természetes evolúciót utánozza [9]. A keresési teret egyedek populációjával reprezentálja. Minden egyed egy-egy lehetséges megoldását jelenti a problémának, városok sorrendjét tartalmazza. Ez a sorrend az utazó ügynök által bejárandó városok sorrendje. A genetikus algoritmus az egyedek keresztezésével, majd szelekciójával állítja elő a populáció egyes generációit.

A genetikus algoritmus pszeudókódja:

- kezdő populáció generálása
- ciklus
- egyedpárok kiválasztása
- gyerek elemek létrehozása az egyedpárokból
- gyerek elemek véletlenszerű mutálása
- populáció kiterjesztése a gyerek elemek hozzáadásával
- kiterjesztett populáció méretének csökkentése szelektálással
- amíg megállási feltétel
- legjobb egyed visszaadása

Először egy kezdő populációt kell generálni. Utána minden iterációban egyedpárokat kell kiválasztani. Ezek az egyedpárok lesznek a szülő egyedek, melyek genotípusát keresztezve jönnek létre a gyerek egyedek. Ezután az új gyerek egyedeket leíró genomot nullához közeli valószínűséggel mutálni kell. Az így létrejövő gyerek egyedeket a populációhoz adva bővíteni kell azt. Majd a célfüggvénynek megfelelő szelekciós művelettel az eredeti méretére kell csökkenteni a populációt. Minden egyes ilyen iteráció a populáció egy generációját állítja elő. A megállási feltétel egy adott generációszám elérése.

4. Keresztező operátor

A genetikus algoritmus egyik kulcsfontosságú része a keresztezés, amely az egyedek genomjait változtatja egyik generációról a másikra. A keresztezés a természetben végbemerő génkeresztezés analógiáját utánozza. Ez egy olyan eljárás, amely két kiválasztott szülő egyedből generál egy gyerek egyedet. A következőekben az ütemezési, illetve a sorrendtervezési feladatoknál alkalmazott keresztező operátorokat mutatjuk be.

4.1. Sorrendkeresztezés (Order Crossover – OX)

Ez a legelső és legfőbb keresztező operátor változat, amelyet az utazó ügynök probléma megoldására alkalmaznak [10]. A leszármazott generálása során először az első szülőből választ egy rész sort, majd a másik szülő génjeinek relatív sorrendjét megőrizve örökíti azt tovább. Példaként vegyünk két szülőegyedet P_1 és P_2-t. A P_1 és P_2 szülőkben jelöljünk ki két vágási pontot (jelölje „|”), miközben a kijelölt szakasz mérete azonos marad.

P_1 = (1 4 | **6 3 2** | 7 5 8)

P_2 = (2 6 | **8 5 1** | 3 4 7)

A két vágási pont közötti részsorok átmásolódnak az O_1 és O_2 leszármazottakba.

O_1 = (_ _ | **6 3 2** | _ _ _)

O_2 = (_ _ | **8 5 1** | _ _ _)

A maradék üres helyeket töltsük fel a másik szülő második vágási pontjától kezdődően azon elemekkel, amelyek még nincsenek a leszármazottban.

A P_2 szülő második vágási pontja utáni elemek sorban a következők:

3 4 7 2 6 8 5 1 .

A 6, 3 és 2 eltávolítása után a másolásra kerülő elemek a következőek:

4 7 8 5 1 .

Ezeket beillesztve az leszármazottba a második vágási pont után a következőt kapjuk:

O_1 = (5 1 | **6 3 2** | 4 7 8).

Ezt az O_2 –re is hasonlóan megtehetjük, amely a következőképpen alakul:

O_2 = (3 2 | **8 5 1** | 7 4 6).

4.2. Az első sorrendkeresztezés operátor variáció (VOX1)

K. Deep és H. Mebrahtu a következő kérdést tette fel: Miért használjuk mind a két szülőben ugyanazokon a helyeken lévő vágási pontokat? [11] Az előzőekben bemutatott operátort alapul véve új keresztező operátorokat dolgoztak ki. Az elsőnél a vágási pontok helyének meghatározásán változtattak, mégpedig azáltal, hogy a két szülőben különböző pozíciókban jelölték ki a vágási pontokat. Ugyanakkor a kijelölt szakasz hossza megegyezik.

Példaként ismét vegyünk két szülőt P_1 és P_2 –t, majd véletlenszerűen és eltérő pozíciókban kijelölünk két-két vágási pontot, miközben a kijelölt szakasz mérete azonos marad.

P_1 = (1 |**4 6 3** |2 7 5 8)

P_2 = (2 68 | **5 1 3** | 4 7).

A vágási pontok közötti részeket másoljuk át a leszármazottakba. Az üres helyeket töltsük fel a másik szülő második vágási pontjától kezdődően azon elemekkel, amelyek még nincsenek a leszármazottban, ezek a következő elemek:

4 7 2 8 5 .

Ezeket beillesztve a leszármazottba a második vágási pont után a következőt kapjuk:

O_1 = (5 | **1 6 3** | 4 7 2 8).

Ezt az O_2 –re is hasonlóan megtehetjük, amely a következőképpen alakul:

O_2 = (8 4**6** | **5 1 3** | 2 7).

4.3. A második sorrendkeresztezés operátor variáció (VOX2)

Ebben az esetben nemcsak a vágási pontok pozíciója különbözik, hanem a vágási pontok által közbezárt sor hossza is. A leszármazott előállításának lépései ebben az esetben is hasonlóak az OX operátornál alkalmazottakhoz.

Az előzőekhez hasonlóan vegyünk P_1 és P_2 szülőt, majd véletlenszerűen kiválasztott és eltérő pozíciókban kijelölünk két-két vágási pontot úgy, hogy közben a közbezárt sor hossza is véletlenszerű legyen.

P_1 = (1 4 6 32| **7 5** | 8)

P_2 = (2 | **6 8 5 1** | 3 4 7).

A vágási pontok közötti részeket másoljuk át a leszármazottakba, majd a második vágási pont után a P_1 szülőből a 3 4 2 6 8 1–et, míg a P_2 szülőből a 3 2 7 4–et.

A leszármazottak a következők lesznek:

O_1 = (4 2 6 8 1 | **7 5** | 3)

O_2 = (4 | **6 8 5 1** | 327).

4.4. Javasolt keresztező operátor

Az előzőekben bemutatott operátorok mindegyike rendelkezik ugyanazzal a közös tulajdonsággal. A szülő vágási pontjai közötti sor a leszármazottban is megtartja abszolút pozícióját. Ezáltal részben rögzülnek az elemek, így kevésbé rendeződik át szabadon az elemek sorrendje a leszármazottban.

Ebből kiindulva jött az ötlet, hogy miért ne legyen véletlenszerűen megválasztva a leszármazottban az a pozíció, ahová a szülőtől örökölt sor kerül. Továbbá az üres helyek feltöltésére a másik szülő sorának kezdőpontjától kezdve történjen. Ezáltal mind a második szülőből feltöltésre kerülő elemek is megőrzik a relatív sorrendjüket. Így egy esetleges optimális szakasz mind a két szülőből továbböröklődhet a gyerekbe, miközben annak pozíciója változik. Mivel az utazó ügynök problémánál egy olyan sor előállítása a feladat, amelynek a kezdő és végpontja ugyanaz, ezáltal a pozíció változás pozitív hatást idézhet elő a leszármazott sorrendjében. Az azonosítás végett nevezzük a javasolt keresztező operátort RPOX operátornak.

Vegyünk két elemet, melyek legyenek P_1 és P_2 szülők, majd jelöljük ki a vágási pontokat.

P_1 = (1 4 | **6 3 2** | 7 5 8)

P_2 = (2 6 8 |**5 1 3 4** | 7).

A két vágási pont közötti részsorok átmásolódnak az O_1és O_2leszármazottakba.

O_1 = (_ _ | **6 3 2** | _ _ _)

O_2 = (_ _ _ | **5 1 3 4** | _).

A maradék üres helyeket töltsük fel a másik szülőben levő elemekkel sorban az elejétől.

A P_2 szülőből átkerülendő elemek sorrendje a 6, 3 és 2 eltávolítása után következő:

8 5 1 4 7 .

Ezeket beillesztve az leszármazottba a következőt kapjuk:

O_1 = (8 5 | **6 3 2** | 1 4 7).

Ezt az O_2 –re is hasonlóan megtehetjük, amely a következőképpen alakul:

O_2 = (6 2 7 | **5 1 3 4** | 8).

5. Teszteredmények

Implementáltuk a genetikus algoritmust OX1, VOX1, VOX2 és RPOX operátorokkal. A VOX1 és VOX2 operátorok esetében törekedtünk arra, hogy a lehető legpontosabban implementáljuk, azonban nem minden részlet volt elérhető hozzá.[11]

Két teszt volt végrehajtva. Az első esetben egy mesterséges feladat lett generálva 128 várossal. A távolságok a városok között 10 és 100 közötti véletlen számokból lettek legenerálva. Ezután egy minimális költségű optimális út lett létrehozva mesterségesen, amelyben a távolságok 1 és 10 közötti véletlen számokból letttk generálva. Mindez azért lett így meghatározva, hogy szignifikáns legyen a különbség a legkisebb költségű út és a többi lehetséges út között. A teszt során a populációszám 256-nak lett választva, míg a generációk száma 32 000-nek.

1. táblázat *Az első teszt eredménye*

generációk	→□	**4000**	**8000**	**12000**	**16000**	**20000**	**24000**	**28000**	**32000**
OX1	best	1134,9	964,0	928,8	879,5	727,2	695,4	695,4	674,2
	worst	1378,2	1160,2	1035,8	956,2	886,7	768,8	768,8	768,8
VOX1	best	827,0	679,8	600,2	574,9	452,4	426,7	423,9	411,1
	worst	960,7	769,8	698,1	684,6	641,2	525,5	525,5	493,9
VOX2	best	799,9	672,1	593,5	557,1	516,7	466,4	436,7	428,9
	worst	938,4	755,3	642,4	584,9	565,5	519,8	463,4	458,7
RPOX	best	777,8	594,0	528,3	487,7	468,1	437,5	409,2	389,8
	worst	800,1	674,1	577,2	556,2	540,9	518,2	516,2	516,2

Az algoritmus többször is futatva volt, majd a legjobb és a legrosszabb eredmények összesítésre kerültek. Az **1. táblázat**ban látható, hogy az RPOX operátort alkalmazva az algoritmus sokkal hamarabb közelít az optimum felé, mint a többi keresztező operátort alkalmazva.

A második teszt során hat benchmark probléma lett választva tesztfeladatnak, a TSPLIB-ből [12]. A teszt során a populáció maximális száma 128, míg a maximális generációszám 30 000 volt. A tábla mutatja a különböző keresztező operátorokat alkalmazó genetikus algoritmusok által elért legjobb és legrosszabb eredményeket.

A **2. táblázat**ban lévő értékek azt mutatják, hogy a VOX1 és VOX2 operátort alkalmazó algoritmusok az optimálishoz (zárójelben) közelebbi eredményeket értek el. Tovább az is látszik, hogy a RPOX operátort alkalmazó algoritmus egy kicsivel még azoknál is jobban teljesített.

2. táblázat *A második teszt eredménye*

	solution	**OX1**	**VOX1**	**VOX2**	**RPOX**
gr48 (5046)	best	6485,0	5207,0	5166,0	5128,0
	worst	7121,0	5486,0	5216,0	5501,0
eli51 (426)	best	441,2	432,0	437,6	430,4
	worst	454,8	439,7	445,9	449,8
eli76 (538)	best	660,9	572,3	569,3	560,1
	worst	708,9	585,1	577,0	585,0
kroA100 (21282)	best	27 738,9	21 795,2	22 169,7	21 602,7
	worst	36 413,7	22 617,0	23 225,2	22 553,9
eli101 (629)	best	752,1	677,0	673,8	668,4
	worst	835,0	698,6	699,3	693,7

6. Következtetések

A cikkben bemutattuk az Utazó ügynök problémájának megoldására alkalmazható genetikus algoritmus keresztező operátorait és egy új operátor alkalmazására is javaslatot tettünk. A keresztező operátorok tulajdonságainak ismertetésén túl rávilágítottunk a jellegzetességeikre. Majd egy olyan operátort mutattunk be, amely a meglévő keresztező operátoroktól eltérő tulajdonsággal rendelkezik. Ezáltal gyorsabb optimum eltérést biztosít, mint a referenciának választott sorrendkeresztező operátor, és legalább annyira hatékony, mint a meglévő variációi ennek az operátornak.

A genetikus algoritmus összetett termelés ütemezési programban való alkalmazásának feltárása és hatékonyságának növelése még nem teljes. A cikkben nem tértünk ki a genetikus algoritmus paraméterei és a szelekciós eljárás által a keresési eredményre gyakorolt hatásra. Ezeknek és a felvetett új operátor további vizsgálata a következő lépés egy összetett ütemező program megalkotásában.

Szakirodalmi hivatkozások

[1] E. D. Kosiba, J. R. Wright And A. E. Cobbs: *Discrete Event Sequencing as a Traveling Salesman Problem.* Computers in Industry, Vol. 19 (1992), No. 3, 317–327.

[2] M. Gendreau, A. Hertz And G. Laporte: *New Insertion and Post-Optimization Procedures for the Traveling Salesman Problem*. Operations Research, Vol. 40 (1992), No. 6, 1086–1094.

[3] D.J. Rosenkrantz, R.E. Sterns And P.M. Lewis: *An Analysis of several Heuristics for*

the Traveling Salesman Problem. SIAM Journal on Computing, Vol. 6 (1977), No. 3, 563–581.

[4] G. Laporte: *The Traveling Salesman Problem: An Overview of Exact and Approximate Algorithms*. European Journal of Operational Research, Vol. 59 (1992), No. 2, 231–247.

[5] J. H. Holland: *Adaptation in Natural and Artificial Systems*: *An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* (U Michigan Press, 1975).

[6] L. Davis: *Applying Adaptive Algorithms to Epistatic Domains*. Proc of the International Joint Conference on Artificial Intelligence – IJCAI85, Vol. 1 (1985), 162–164.

[7] Whitley, Darrell, Timothy Starkweather, and Dan Shaner. *The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination*. Colorado State University, Department of Computer Science, (1991)

[8] Lenstra, Jan Karel, Ahg Rinnooy Kan, And David B. Shmoys. *The traveling salesman problem: a guided tour of combinatorial optimization*. Vol. 3. Chichester: Wiley, (1985)

[9] D.E. Goldberg: *Genetic Algorithms in Search*. Optimization and Machine Learning. (Addison-Wesley, 1989).

[10] Davis, L.: *Applying Adaptive Algorithms to Epistactic Domains*. Proceedings of the International Joint Conference on Artificial Intelligence – IJCAI85, Vol. 1 (1985) 162–164.

[11] K. Deep, H. Membrahtu: *New Variations of Order Crossover for Travelling Salesman Problem.* International Journal of Combinatorial Optimization Problems and Informatics, Vol. 2 (2011), No.1, 2–13.

[12] http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ (megtekintve: 2013. November)