EGY HEURISZTIKUS MÓDSZER A KVADRATIKUS HOZZÁRENDELÉSI PROBLÉMA KÖZELÍTŐ MEGOLDÁSÁHOZ [1]

BORGULYA ISTVÁN
PTE Közgazdaságtudományi Kar

Tanulmányunkban egy evolúciós algoritmust ismertetünk a kvadratikus hozzárendelési probléma megoldására. Az algoritmus működése három, egymás utáni fázisra bontható: egy előkészítő, valamint két különböző helyi kereső fázisra. Az első fázisban a kezdő populáció minőségét javítja. A második fázisban egy helyi kereső eljárással felváltva javít minden megoldást (egyedet), és közben periodikusan újabb megoldásokat generál a meglévők környezetében. A harmadik fázisban a helyi kereső eljárással folytatja a megoldások javítását, és közben a legrosszabb megoldásokat periodikusan újakra cseréli. Az algoritmust a QAPLIB könyvtár jól ismert tesztfeladataival ellenőriztük. Az algoritmus a feladatok 98%-nál megtalálta a legjobb ismert megoldásokat, vagy a legjobb ismert megoldás 1%-os környezetébe került.

1 Bevezetés

A kvadratikus hozzárendelési probléma (QAP), egy klasszikus kombinatorikai optimalizációs probléma. NP-teljes feladat, és egzakt megoldása csak kisméretű feladatok eseten lehetséges, ill. gazdaságos.

A probléma a következő: adott n objektum, melyet n különböző helyre kell elhelyezni. Tudjuk, hogy f_{ij} érték áramlik az i és j objektum közt, és a k és l hely közti távolság pedig d_{kl}. Az objektumok olyan elrendezését keressük, amelynél az érték $\times$ távolság szorzatok összege minimális. Matematikailag a következőképpen formalizálhatjuk: legyen $F = \{f_{ij}\}$ és $D = \{d_{kl}\}$, két $n \times n$-es mátrix, és keresünk egy olyan π^* permutációt, amelyre minimális

$$Z(\pi) = \min_{\pi \in \Pi(n)} \sum_{i=1}^{n} \sum_{i=j}^{n} f_{ij} d_{\pi_i \pi_j}$$

ahol $\Pi(n)$ n elem permutációinak halmaza és π_i, π_j a $\pi \in \Pi(n)$ permutáció i-edik és j-edik pozíciójának értékét jelöli.

A QAP alkalmazásával számos mérnöki, ill. tudományos feladatban találkozunk. Néhány gyakorlati probléma ezek közül, pl. egyetemtervezési probléma; irodák, szobák elrendezése irodaházban, ill. korházban; VLSI lapocskák elhelyezése az alaplapon (részletesebben: [5], [8], [15]).

[1] Beérkezett: 2001. december 8.

A QAP egzakt megoldására a dinamikus programozás, különböző metszési módszerek és a korlátozás és szétválasztás módszere alkalmazhatók. $n = 20$-nál nagyobb méretű feladatok esetén az egzakt megoldást azonban nem tudjuk meghatározni elfogadható időn belül [5]. Mivel a legtöbb gyakorlati feladat nagyméretű probléma, így számos heurisztikát alkalmaznak, melyek elfogadható időn belül jó megoldást találnak.

Sokféle heurisztika létezik a QAP megoldására. A legismertebb csoportjaik a következők: konstrukciós módszerek, javító módszerek, szimulált hűtés (SA), tabukeresés (TS), evolúciós algoritmus (EA) változatok (pl. genetikus algoritmus (GA), genetikus programozás (GP), evolúciós stratégia (ES)), hangya kolóniák (részletesebben, pl. [8], [4], [15], [10], [12], [11]). Egyes heurisztikus módszerek gyorsasága és pontossága tovább javítható, pl. különböző optimalizáló, vagy párhozamos számításokkal. Így Ahuja et al. [1] a GA teljesítményét egyes műveletek optimalizálásával, több populáció párhuzamos alkalmazásával javítja; Talbi et al. [17] párhuzamos TS algoritmust fejleszt, amely nagyméretű feladatok esetén is nagyon jó közelítést nyújt az optimumra; Bölte et al. [3] GP segítségével állít elő jobb "hő ütemező" függvényt az SA módszerhez. E mellett számos más elven működő heurisztikus módszer alkalmazható. A módszerek közt Boltzmann gép, vagy klaszterező módszerek szintén találhatók.

A heurisztikus módszerek hatékonyságát több összehasonlító tanulmány vizsgálja (pl. [13], [16], [11]). Általában megállapítható, hogy a módszerek a legjobb ismert megoldást 1%-on belüli pontossággal képesek közelíti; egyes feladatokat más-más módszerek oldanak meg sikeresebben és a módszerek egy része csak $n < 100$ méretű problémát tud eredményesen kezelni. A legeredményesebb heurisztikák a TS, SA és a "helyi javító" módszer ([1], [7]).

E heurisztikus módszerek körét egy újabb algoritmussal kívánjuk bővíteni. Célunk, hogy az algoritmus

a) kis, közepes, vagy nagyméretű problémáknál egyaránt alkalmazható legyen;

b) PC-n a szokásos feladatok esetén, elfogadható időben nyújtson jó közelítést (a legjobb ismert megoldást 1%-on belüli pontossággal közelítse); és

c) egyszerűen, néhány paraméter beállításával legyen kezelhető.

Az algoritmust a QAPLIB könyvtár [6] tesztfeladataival ellenőriztük. A feladatok 98%-át sikeresen oldotta meg: vagy megtalálta a legjobb ismert megoldást, vagy a legjobb ismert megoldást 1%-on belüli pontossággal közelítette meg. A feladatok további 2%-ban 1-1.5% pontossággal közelítette a legjobb ismert megoldást. Összehasonlításra a greedy genetikus algoritmust [1] választottuk. Megállapítható, hogy új módszerünk PC-n, általában rövidebb idő alatt, közel azonos pontosságú közelítést talált, mint a greedy genetikus algoritmus.

A továbbiakban nézzük először az új algoritmus leírását, majd a tesztfeladatok megoldását, melyek az alkalmazás lehetőségét szemléltetik.

2 Az algoritmus alapgondolata

Az algoritmus, nevezzük EF_QAP-nek (Evolutionary Framework for QAP), egy korábbi klaszter-alapú optimalizációs algoritmus [2] módosításával jött létre. Egy olyan hibrid EA, amely az evolúciós ciklust egy helyi kereső eljárással bővíti. Új algoritmusunk azonban a konvergencia gyorsítása és minél jobb minőségű eredmények elérése érdekében, a korábbi hibrid EA-któl eltérően nem egy, hanem 3 fázisból áll. A három fázis mindegyike egy-egy EA, amelyeket egymás után kell végrehajtani. Az első fázis egy "előkészítő" fázis, amely a kezdő populáció minőségét kívánja javítani. A második fázis olyan hibrid EA, amely keresés közben folyamatosan növeli a populáció méretét. E méret növeléssel gyorsítani kívánja a konvergenciát. Végül a harmadik fázis folytatja a megoldások javítását. Eltérően a második fázistól itt állandó a populáció mérete, viszont a legrosszabb megoldásokat periodikusan újakra cseréli az algoritmus.

Az algoritmus tehát három EA-ból áll. Tekintsük a permutációkat egyedeknek (vagy megoldásoknak). Minden EA-ban generációnként (iterációnként) egy szülőből másolással egy utódot hozunk létre. Mutációt (néhány párcsere a permutációban + helyi kereső eljárás) alkalmazunk, majd az utódot a leghasonlóbb egyeddel (vagy magával a szülővel) párba állítva szelektálunk az egyed és az utód közt (crowding selection). Jósági függvénynek (fitness function) a Z függvényt választhatjuk. Az EF_QAP működését meghatározó három EA feladata részletesebben a következő:

1. Az első fázisban az algoritmus először véletlenszerűen generálja a kezdő populáció egyedeit. Utána véletlenszerűen generált utódokkal próbálja javítani őket. Egy utód mindig csak a leghasonlóbb korábbi megoldást javíthatja.

2. A második fázisban a megoldások minőségét mutációval, azaz egy összetett helyi kereső eljárással javítja. Az eljárás a megoldások javításához a megoldások környezetében olyan utódokat választ, melyek a korábbi megoldásból (permutációból) 1-2 párcserével és egy szomszédságban kereső eljárással előállíthatók (A mutáció tehát az *1-2 párcsere + szomszédságban kereső eljárás* transzformációval állítja elő a szülőből az utódot). A javításra kerülő megoldás kiválasztásában prioritást élvez a legjobb, legkisebb Z függvényértékű megoldás. 0.5 valószínűséggel a legjobb, $0.5/t$ valószínűséggel (ahol t a populáció mérete) pedig tetszőleges megoldást választ az algoritmus.

   A helyi kereső eljárás tehát a véletlenszerűen választott egy-két párcsere után egy *szekvenciális, szomszédságban kereső eljárást* alkalmaz, amely végigvizsgálja az új elem szomszédos permutációit. E szekvenciális, szomszédságban kereső eljárás (sequential neighbourhood search, [8]) a permutáció pozícióin a következő sorrendben alkalmaz párcseréket:

$$(1,2),(1,3),\ldots,(1,n),(2,3),\ldots,(n-1,n),(1,2),\ldots$$

Az algoritmus a szomszédságban kereső eljárást mindaddig alkalmazza, amíg lehetséges az eredmény javítása. Ha a keresés végére ért, és keresés közben valamely párcsere javított a helyi optimum közelítés értékén, újból kezdi a szomszédságban kereső eljárást. Emellett, hogy a végrehajtási időt gyorsítsa, a függvényérték számítást Taillard [16] módszerével szintén egyszerűsíti. Ehhez az i és j objektumok cseréjénél először csak a függvényérték változását számolja ki

$$\begin{aligned}\Delta Z = {} & f_{ii}(d_{\pi_j\pi_j} - d_{\pi_i\pi_i}) + f_{ij}(d_{\pi_j\pi_i} - d_{\pi_i\pi_j}) + \\ & + f_{ji}(d_{\pi_i\pi_i} - d_{\pi_j\pi_j}) + f_{jj}(d_{\pi_i\pi_i} - d_{\pi_j\pi_j}) + \\ & + \sum_{\substack{k=1\ldots n \\ k\neq i,j}} \Big(f_{ki}(d_{\pi_k\pi_j} - d_{\pi_k\pi_i}) + f_{kj}(d_{\pi_k\pi_i} - d_{\pi_k\pi_j}) + \\ & + f_{jk}(d_{\pi_j\pi_k} - d_{\pi_i\pi_k}) + f_{ik}(d_{\pi_j\pi_k} - d_{\pi_j\pi_k}) \Big)\end{aligned}$$

Ha a $\Delta Z < 0$, akkor $-\Delta Z$-vel javul a függvény értéke. Az algoritmus a párcserét ténylegesen csak abban az esetben hajtja végre, ha javul a függvény értéke.

Általában a korábbi megoldásból az 1-2 párcsere + szomszédságban kereső eljárás transzformációval egyre jobb megoldást kapunk. Egyes feladatoknál a továbblépéshez azonban az 1-2 párcsere nem mindig elegendő, az algoritmus valamely lokális optimumnál "elakad". A megoldások javítása érdekében ezért újabb megoldásokat generál az algoritmus. Egy újabb megoldást valamely már létező megoldásból véletlenszerűen, maximum $n/8$ számú párcsere + szomszédságban kereső eljárás transzformációval állít elő. Így az új megoldások a meglévők "közelében" keletkeznek, és mint újabb változatok, növelik annak az esélyét, hogy az algoritmus közelíteni tudjon a globális optimumhoz. Az algoritmus ezért a második fázisban periodikusan, mindaddig újabb megoldásokat generál a meglévők környezetében, amíg a megoldások (egyedek) száma egy maximális értéket el nem ér.

3. A harmadik fázisban az algoritmus folytatja a második fázis helyi kereső eljárásával a megoldások javítását. Szintén generál újabb megoldásokat, de lecseréli velük a legrosszabb megoldásokat. Legrosszabbnak a legnagyobb célfüggvény értékű megoldásokat tekinti az algoritmus, és periódusonként adott százalékukat cseréli le.

3 Az algoritmus

Jelölések

Vezessük be a következő jelöléseket:

- Jelöljük az egyes EA-kat EA1, EA2 és EA3-al.

- Ugyanazon populációt alkalmazza mindhárom EA. Legyenek a $p_1, \ldots, p_t$ permutációk az egyedek (megoldások). Jelölje az it-edik generáció populációját $P(it)$, és az i-edik egyedet p_i. A jósági függvény legyen azonos a Z függvénnyel.

- Jelölje `Csere(q)` a párcsere eljárás-t. Az eljárás a q permutációban véletlenszerűen kicseréli két pozíció értékét.

- Jelölje `Nhs(q)` a szekvenciális, szomszédságban kereső eljárást, amely a q utód szomszédos permutációit mindaddig ismételten végigvizsgálja, amíg a megoldás értékén lehetséges javítani. Ha tud javítani, minden alkalommal végrehajtja a párcserét a q megoldásban.

- Jelölje `Ujegyed` az új megoldást generáló eljárást. Az eljárás úgy generál egy új q megoldást, hogy véletlenszerűen választ egy p_i-t ($p_i \neq 0$, $i \in \{1, 2, \ldots, t\}$), átmásolja q-ba, alkalmazza rá maximum $n/8$-szor a `Csere(q)` eljárást, majd az `Nhs(q)` eljárást. q a $t+1$-edik egyed lesz.

- Jelölje `SortDel` a rendező eljárást, amely a prototípusokat a Z függvényértékek szerint növekvő sorba rendezi, törli a legrosszabb megoldások *ddp* százalékát, és helyettük új megoldásokat generál.

- Az x és z permutáció hasonlóságának mértékét $H(x,z) = 1/(1+d(x,z))$ határozza meg, ahol $d(x,z)$ a két permutáció Hamming-távolsága.

- Jelölje a véletlenszám generátort `Rnd` (egyenletes eloszlás $(0,1)$-en).

Paraméterek

7 paraméter befolyásolja az algoritmus futását: *tmax*, *t*, *itt*, *kn*, *ddp*, *m* és *timeend*. Szerepük a következő:

- *tmax* - a populáció maximális mérete.

- *t* - a populáció maximális mérete az első fázisban.

- *itt* - az 1. fázis paramétere. Ha az *it* iteráció szám nagyobb, mint *itt*, megkezdődik a 2. fázis.

- *kn* - az ellenőrzések időpontját meghatározó paraméter. A legjobb megoldások megkeresése, vagy a legrosszabb megoldások törlése, újakra cserélése csak minden *kn*-dik iterációban történik.

- *m* - a 2. fázis paramétere. A 2. fázisban az algoritmus minden *m*-dik iterációban növeli a populáció méretét.

- *ddp* - a 3. fázis paramétere. A legrosszabb megoldások *ddp* százalékát törli a 3. fázisban.

- *timeend* - a megállási feltétel paramétere. Az eljárás befejeződik, ha a futási idő (CPU idő másodpercben) nagyobb, mint *timeend*.

Az algoritmus lépései

Procedure EF_QAP($tmax, t, itt, kn, ddp, m, timeend, opt, optp$)

** EA1 **
$it := 0$, $glob := 1$. /* Kezdőértékek beállítása
Legyen $p_i \in \Pi(n)$ $(i = 1, \ldots, t)$, $P(it) \leftarrow \{p_1, \ldots, p_t\}$ /*Kezdő populáció
$Z(p_1), \ldots, Z(p_t)$ kiszámítása
Repeat
  $i := [t * \texttt{Rnd}] + 1$, $q := p_i$ /* Utód generálás
  For $j := 1$ **to** $[n/2]$ **do** Csere(q) **od** /*Mutáció
  $Z(q)$ kiszámítása
  Legyen $H(q, p_z) = \max_j H(q, p_j)$; $j, z \in \{1, 2, \ldots, t\}$. /*Hasonlóság vizsgálat
  If $Z(q) < Z(p_z)$ **then** $p_z := q$ **fi** /*Szelekció
  $it := it + 1$, $P(it) \leftarrow P(it - 1)$
until $itt < it$ /* 1. fázis ismétlési feltétele

** EA2 **
Repeat
  Repeat
    If $0.5 <$ Rnd **then** $i := glob$ **else** $i := [t * \texttt{Rnd}] + 1$ **fi** /* Utód generálás
    $q := p_i$
    For $j := 1$ **to** $[\texttt{Rnd} * 2] + 1$ **do** Csere(q) **od** Nhs(q) /*Mutáció
    $Z(q)$ kiszámítása
    If $Z(q) < Z(p_i)$ **then** $p_i := q$ **fi** /*Szelekció
    $it := it + 1$, $P(it) \leftarrow P(it - 1)$
    If (it mod m) $= 0$ **then Újegyed fi** /* Az egyedek számának növelése
  until (it mod kn) $\neq 0$
  Legyen $Z(p_i) = \min_j Z(p_j)$, $i, j \in \{1, 2, \ldots, t\}$, $glob := i$ /*Legjobb közelítés kiválasztása
  $opt = Z(glob)$, $optp = p_{glob}$
  If "futási idő" $> timeend$ **then exit fi** /* Megállási feltétel
until $t < tmax$ /* 2. fázis ismétlési feltétele

** EA3 **
Repeat
  Repeat
    $i := [t * \texttt{Rnd}] + 1$, $q := p_i$ /* Utód generálás
    For $j := 1$ **to** $[\texttt{Rnd} * 2] + 1$ **do** Csere(q) **od**, Nhs(q) /* Mutáció
    $Z(q)$ kiszámítása
    If $Z(q) < Z(p_i)$ **then** $p_i := q$ **fi** /* Szelekció
    $it := it + 1$, $P(it) \leftarrow P(it - 1)$
  until (it mod kn) $\neq 0$ /* Ismétlési feltétel
  SortDel /* Legrosszabb egyedek újracserélése
  Legyen $Z(p_i) = min_j Z(p_j)$, $i, j \in \{1, 2, \ldots, t\}$, $glob := i$ /* Legjobb közelítés kiválasztása
  $opt = Z(glob)$, $optp = p_{glob}$
until "futási idő" $> timeend$ /* Megállási feltétel
exit
end

4 Tesztproblémák megoldása

4.1 Paraméterek megválasztása

QAPLIB könyvtár[2] feladatainak megoldásához először néhány nehezebb tesztproblémát választottunk ki, amelyeknél a konvergencia sebességét, a megoldás

[2] http://fmatbhp1.tu.graz.ac.at/~karisch/qaplib

minőségét a paraméter értékek függvényében vizsgáltuk. A megfelelő paraméter értékek megválasztásához minden paraméter szerepét külön vizsgáltuk, tanulmányoztuk külön-külön a három fázis szerepét, és végül a legjobb paraméter-kombináció kiválasztását kíséreltük meg.

E körültekintő folyamat végeredményét a következőképpen foglalhatjuk össze:

- Az első fázis szerepe fontos, javítja a megoldások minőségét. Általában kisebb szórással kapjuk a végeredményeket e fázis alkalmazása esetén. Általában 400 véletlenszerűen generált megoldás elegendő a kezdő populáció minőségének javításához ($itt = 400$).

- A második fázis gyorsítja a konvergenciát. $t = 10$ kezdőpopuláció méret alkalmazásával a tesztproblémák többségénél a leggyorsabb, és legjobb közelítéseket lehet elérni. (A $t = 2$ érték az esetek 30%-ban nagyon jó eredmény nyújt, a többiben a $t = 10$ megfelelőbb.)

- Az m és kn paraméterek, amelyek a populáció méret növelés és az ellenőrzések időpontját határozzák meg, függnek a dimenziótól. Értékeik azonosak lehetnek és kis, közepes és nagy dimenziók esetén (10-30, 31-89, és 90-) rendre 50, 25 és 5 értékek a megfelelőek.

- A harmadik fázisban a populáció mérete szintén dimenziófüggő. Kis, közepes és nagy dimenziónál rendre 30, 30 (vagy 60) és 90 elemű populációt ($tmax$) célszerű alkalmazni. A ddp paraméter értéke feladattól függetlenül 0.3-nak választható.

- A futásidők feladatfüggők. A különböző nehézségű feladatok megoldásához ugyanazon dimenzió esetén is más-más futásidők szükségesek a megkívánt pontosság eléréséhez. Így minden feladatnál külön kell a futásidőt ($timeend$) becsülni.

E dimenziótól függő, háromféle paraméterkombinációt alkalmaztuk tehát a QAPLIB feladatainak megoldásánál:

kis dimenzióknál:
$t = 10, \quad tmax = 30, \quad m = 50, \quad kn = 50, \quad ddp = 0.3, \quad itt = 400;$
közepes dimenzióknál:
$t = 10, \quad tmax = 30, \quad m = 25, \quad kn = 25, \quad ddp = 0.3, \quad itt = 400;$
nagy dimenzióknál:
$t = 10, \quad tmax = 90, \quad m = 5, \quad kn = 5, \quad ddp = 0.3 \quad itt = 400.$

Néhány nehéz feladat esetében (3%-a a feladatoknak), amelyeknél csak 1-1.5%-os pontossággal tudtuk a legjobb ismert megoldást közelíteni, a $t = tmax = 60$ paraméterértékek megfelelő eredményeket adtak.

4.2 Teszteredmények

Algoritmusunk tehát a QAPLIB könyvtár [6] tesztproblémáival ellenőriztük. EF_QAP a problémák 98%-át sikeresen oldotta meg; a legjobb ismert megoldást 1%-on belüli pontossággal közelítette meg, vagy megtalálta az ismert legjobb megoldást.

Bemutatásra három csoportot válogattunk a problémák közül. Arra törekedtünk, hogy eredményeink összehasonlíthatók legyenek más módszer(ek) eredményeivel, és különböző dimenziójú, típusú feladatok egyaránt szerepeljenek közöttük. Az első csoport egy ilyen válogatás (Bölte et al. [3] 14 válogatott tesztproblémája), a második csoport a QAPLIB további 100-256 dimenziós problémái és a harmadik csoport, néhány időigényesebb probléma, melyet sikeresen oldott meg az EF_QAP.

Összehasonlításhoz Ahuja et al. [1] greedy GA módszerét (jelölés: gGA) választottuk, mellyel a QAPLIB maximum 100 dimenziós problémáinak túlnyomó részét megoldották (100 dimenzió feletti problémák megoldására nem alkalmas a gGA). A gGA-t C-ben programozták és egy HP 6000 számítógépen futtatták. Mi az EF_QAP-t Visual Basic-ben programoztuk és egy HP Brio (400 Mhz) számítógépen futtattuk. Bár a mi algoritmusunk lassabban futott, az időeredmények mégis lehetővé tették a módszerek összehasonlítását.

A 14 válogatott tesztprobléma eredményét az *1. táblázat*, a további, magasabb dimenziós problémák eredményét a *2. táblázat*, az "időigényesebb" problémák eredményeit pedig a *3. táblázat* tartalmazza. Mindhárom táblánál a bemutatott eredmények több futtatás átlagos eredményei: a 70 dimenzió alatti problémákat 10-szer, a magasabb dimenziósakat pedig 5-ször futtattuk le. Az egyes problémák futási idejét a *timeend* paraméterrel adtuk meg. (Egy probléma minden egyes futásánál eredménynek a végső pontosságot és a pontosság eléréséhez szükséges futási időt tekintettük). A táblákban megadtuk a probléma nevét és dimenzióját, az ismert legjobb megoldást, az ismert legjobb megoldástól való átlagos elétérést százalékban (hiba), az átlagos futási időt (CPUsec), a *timeend* paraméter értékét és az ismert legjobb megoldás 1%-os környezetébe eső megoldások százalékát (Opt1%). Az EF_QAP és a gGA összehasonlításához a gGA adatait is megadtuk (amennyiben ismertek).

Név	Dim	Ismert legjobb	Hiba (%)		CPUsec			$Opt_{1\%}$
			gGA	EF_QAP	gGA	EF_QAP	*timeend*	EF_QAP
Nug20	20	2670	0.00	0.00	48.9	10.3	12	100
Nug30	30	6124	0.07	0.11	177.1	73.6	105	100
Tho30	30	149936	0.00	0.27	197.8	80.2	150	100
Tho40	40	240516	0.32	0.28	479.0	129.7	150	100
Sko42	42	15812	0.25	0.20	503.1	302.7	500	100
Sko49	49	23386	0.21	0.29	626.1	338.0	500	100
Wil50	50	48816	0.07	0.10	1057.6	430.2	650	100
Sko56	56	34458	0.02	0.15	1488.0	1077.0	1350	100
Sko64	64	48498	0.22	0.36	1894.1	1654.0	1800	100
Sko72	72	66256	0.29	0.23	2539.0	1383.0	1800	100
Sko81	81	90998	0.20	0.33	5482.1	4789.0	5000	100
Sko90	90	115534	0.27	0.34	6348.9	6336.0	6500	100

1. táblázat. Válogatott tesztproblémák

Név	Dim	Ismert legjobb	Hiba (%)		CPUsec			$Opt_{1\%}$
			gGA	EF_QAP	gGA	EF_QAP	*timeend*	EF_QAP
Sko100a	100	152002	0.21	0.31	16608	2348	2500	100
Sko100b	100	153890	0.14	0.14	14735	1829	2000	100
Sko100c	100	147862	0.20	0.18	20314	3818	4200	100
Sko100d	100	149570	0.17	0.34	20302	7751	9000	100
Sko100e	100	149150	0.24	0.30	20127	7942	9000	100
Sko100f	100	149036	0.29	0.46	20479	6112	7500	100
Wil100	100	273038	0.20	0.26	20544	4525	8000	100
Tai100a	100	21125314		0.98		16328	20000	100
Tai100b	100	1185996137		0.88		5449	6000	100
Esc128	128	64		0.00		359	1200	100
Tho150	150	8133484		0.43		23564	26000	100
Tai150b	150	498896643		0.80		11809	13000	100
Tai256c	256	44759294		0.14		35458	36000	100

2. táblázat. Magasabb dimenziós tesztproblémák

Név	Dim	Ismert legjobb	Hiba (%)		CPUsec			$Opt_{1\%}$
			gGA	EF_QAP	gGA	EF_QAP	*timeend*	EF_QAP
Chr20b	20	2298	5.13	0.50	96	279	450	80
Ste36a	36	9526	0.27	0.06	710	690	1000	100
Lipa50a	50	62093	0.95	0.68	15486	678	1200	100
Lipa50b	50	1210244	0.00	0.00	1509	282	500	100
Lipa60a	60	107218	0.77	0.79	3057	1518	2000	100
Lipa60b	60	2520135	0.00	0.00	3047	1289	2000	100
Lipa70a	70	169755	0.71	0.76	6148	1757	2000	100
Lipa70b	70	4603200	0.00	0.00	6122	3026	5000	100
Lipa80a	80	253195	0.61	0.59	9518	3600	5000	100
Lipa80b	80	7763962	0.00	0.00	94989	4463	7000	100
Lipa90a	90	360630	0.58	0.56	12358	4123	6500	100
Lipa90b	90	12490441	0.00	0.00	12319	8147	14000	100
Tai50a	50	4941410		1.07		1490	3000	50
Tai60a	60	7208572		0.85		6948	7200	100
Tai80a	80	13557864		1.00		8862	14000	60

3. táblázat. Időigényesebb tesztproblémák eredményei

Az 1. táblázat eredményeit vizsgálva megállapítható, hogy az EF_QAP gyorsabb a gGA-nál. Általában az EF_QAP 2-3-szor hamarabb találja meg a hasonló pontosságú megoldást (0.1%-os az eltérés a két módszer eredményeiben). A megoldás menetét vizsgálva megállapítható, hogy a kívánt pontosságot már a második fázisban, néhány újabb megoldás generálásával, gyorsan eléri az algoritmus.

A 2. táblázat eredményei hasonlóak az előzőhöz: az EF_QAP ismét gyorsabban találta meg a hasonló pontosságú eredményt (A QAPLIB magasabb dimenziós problémáinak csak a felénél ismertek a gGA eredményei). A megoldás menete ugyancsak hasonló az előző feladatokéhoz; itt is a kívánt pontosságot már a második fázisban elérte az algoritmus, nem volt szükség a harmadik fázis helyi keresésére.

A 3. táblázat azon feladatokat, feladatcsoportokat gyűjti össze, amelyeket az EF_QAP algoritmus átlagosnál hosszabb idő alatt oldott meg. Az időadatokat összehasonlítva ennek ellenére most is az EF_QAP bizonyult gyorsabbnak, csak most nagyobb a szórás az egyes módszerek időadatai közt. E

feladatoknál minden esetben szükség volt a harmadik fázis helyi keresésére is. Ilyen problémák pl. az Ste36a, Lipa50a, Lipa50b, ..., Lipa90a, Lipa90b voltak.

Az időigényesebb problémák egy kis részénél viszont a második fázis "gyors" helyi keresése bizonyult eredménytelennek (azaz, a szokásos módszerekhez képest csak lassabban tudta a globális optimumot közelíteni). Feltételezve azt, hogy a helyi optimumok értékei alig különböznek egymástól, sikerült e feladatokat is gyorsabban megoldani. E feladatoknál az első fázisban mindjárt a maximális populáció méretet alkalmaztuk ($t = tmax$). Evvel automatikusan kihagytuk a második fázist, és a harmadik fázis helyi keresése már kellő gyorsasággal adta a kívánt eredményt. Ilyen problémák, pl. tai50a, tai60a, tai80a voltak. E tábla problémái közt találjuk ugyancsak azt a három feladatot (chr20b, tai50a és tai80a), amelyeket nem tudott az algoritmus minden esetben kellő pontossággal megoldani.

A QAPLIB további feladatait szintén sikerrel oldotta meg az EF_QAP. Minden feladatot átlagosan 0.00 (vagy nulla) hibával oldott meg és általában 2-3-szor hamarabb találta meg a gGA-hoz képest a hasonló pontosságú megoldást.

Összességében megállapítható, hogy az EF_QAP a QAPLIB tesztproblémáit meg tudja oldani, és három probléma kivételével, a megoldások mindig az ismert legjobb megoldás 1%-os környezetébe esnek. A futási idők összehasonlítása igen nehéz különböző gépek és programozási nyelvek esetén, de kísérletet tettünk a gGA-nál publikált futási idők [1] összehasonlítására. Ezek alapján az EF_QAP általában 2-3-szor gyorsabban találja meg a hasonló pontosságú eredményt, mint a gGA.

5 Összefoglalás

Az EF_QAP egy háromfázisú evolúciós algoritmus a QAP megoldására. Olyan heurisztika, amellyel kis, közepes és nagyméretű feladat egyaránt megoldható. Az algoritmus a QAPLIB könyvtár jól ismert teszt problémáit sikeresen oldotta meg, a feladatok 98%-ban az ismert legjobb megoldás 1%-os környezetébe került, vagy megtalálta az ismert legjobb megoldást. Összehasonlítva egy másik heurisztikus módszerrel, Ahuja et al. [1] gGA módszerével, megállapíthatjuk, hogy az EF_QAP, a 100 dimenziósnál nem nagyobb feladok esetén, 2-3-szor rövidebb idő alatt nyújtja a hasonló pontosságú eredményeket.

Az algoritmus működését 7 paraméter befolyásolja, de többségük a különböző feladatoknál azonos, rögzített értékkel használható. Az algoritmus működését vizsgálva megállapítható, hogy a tesztproblémák esetén az ismert legjobb megoldást 1-2%-os pontossággal viszonylag hamar meg tudja közelíteni. Utána fokozatosan, egyre lassabban javítja a közelítéseket. A konvergencia gyorsítását az algoritmus első és második fázisa segíti. A gyorsaság szempontjából különösen a 2. fázis fontos: az újabb közelítések generálásával lényegesen meg tudja növelni a keresés gyorsaságát és hatékonyságát.

Köszönetnyilvánítás

A kutatás az OTKA T 030861 támogatásával készült.

Irodalom

1. Ahuja K. R., Orlin J. M., Tiwari A.: A greedy genetic algorithm for the quadratic assignment problem. *Computer & Operations Research* 27 (2000) 917–934.
2. Borgulya I.: Constrained optimization using a clustering algorithm. *Central European Journal of Operations Research.* 8 (1) (2000) 13–34.
3. Bölte A., Thonemann U. W.: Optimizing simulated annealing schedules with genetic programming. *European Journal of Operational Research.* 92 (1996) 402–416.
4. Burkard R. E., Rendl F.: A thermodynamically motivated simulation procedure for combinatorial optimization problems. European Journal of Operations Research, 17(2) (1984) 169–174.
5. Burkard R. E.: Locations with spatial interactions: the quadratic assignment problem. In: Mirchandani P. B., Francis R. L.(eds), *Discrete Location Theory.* Wiley, Berlin, 1991
6. Burkard R. E., Karisch S. E., Rendl F.: QAPLIB - A quadratic assignment library. *Journal of Global Optimization.* 10 (1997) 391–403.
7. Burkard R. E., Çela E., Pardalos P. M., Pitsoulis L. S.: The Quadratic Assignment Problem. In: Pardalos P. M., Du D. Z. (eds): *Handbook of Combinatorial Optimization* Vol. 3 Kluwer Academic Publishers, Dordrecht 1998, pp. 241–339.
8. Çela E.: *The Quadratic Assignment Problem: Theory and Algorithms.* Kluwer Academic Publishers, Dordrecht, 1998.
9. Connolly D. T.: An improved annealing scheme for the QAP. *European Journal of Operational Research.* 46 (1990) 93–100.
10. Fleurent C., Ferland J. A.: *Genetic Hybrids for the Quadratic Assignment problem*, DIMACS Series in Mathematics and Theorical Computer Science, 16 (1994) 173–187.
11. Gambardella L. M., Taillard E. D., Dorigo M.: Ant Colonies for the QAP. *Journal of Operation Research Society* 50 (1999) 167–176.
12. Li Y., Pardalos P. M., Resende M. G. C.: A greedy randomized adaptive search procedure for the quadratic assignment problem. In Pardalos P. M., Wolkowicz H. (eds): *Quadratic assignment and related problems*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 16 (1994) 237–261
13. Maniezzo V., Dorigo M., Colorni A.: Algodesk: An experimental comparison of eight evolutionary heuristics applied to the Quadratic Assignment Problem. *European Journal of Operational Research.* 81 (1995) 188–204.
14. Nissen V.: Quadratic assignment, In: Bäck T., Fogel D. B., Michalewicz Z.: *Handbook of Evolutionary Computation.* Oxford University Press, 1997. pp. G9.10:1–9.
15. Pardalos P. M., Rendl F., Wolkowicz H.: The quadratic assignment problem: a survey of recent developments. In Pardalos P. M., Wolkowicz H.(eds): *Quadratic assignment and related problems*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 16 (1994) 1–42.

16. Taillard D.: Comparison of iterative searches for the quadratic assingnment problem. *Location Science*, 3 (1995) 87–105.

17. Talbi E. G., Hafidi Z., Geib J-M.: *Parallel adaptive tabu search for large optimization problems.* Research report LIFL URA 369 CNRS, University of Lille, March 1997.

A HEURISTIC ALGORITHM FOR THE QUADRATIC ASSIGNMENT PROBLEM

In this paper we describe a new evolutionary algorithm for the Quadratic Assignment Problem. This method can be divided into three stages: a preparatory and two local search stages. The first stage improves the quality of the initial population. The second stage improves the quality of the solutions with a local search procedure while periodically generating new solutions. In the third stage the algorithm continues the application of the local search procedure and replaces the weakest solutions with new ones. We tested our algorithm on all the benchmark problems of QAPLIB. The algorithm managed to find solutions which are either best known or within 1% of the best known solutions for 98% of all tasks.