Hatékony útvonal tervezési algoritmusok

Szabó Ádám

Tartalomjegyzék

1. Útvonal tervezés és nehézségei

1.1. Az útvonal tervezési alapfeladat

Klasszikus példa az útvonal tervezési problémára, amikor adott egy utazóügynök és városok egy diszkrét halmaza, amiket meg kell látogatnia pontosan egyszer tetszőleges sorrendben úgy, hogy a megtett összes út minimális legyen és végül a kiindulási városba térjen vissza. Ezt a feladatot hívják utazó ügynök problémának más néven TSP-nek (Traveling Salesman Problem). A feladat reprezentálható egy teljes, súlyozott (irányított és szimmetrikus) gráffal (G). A gráf csúcsai a városok (V), az élei pedig a városokat összekötő utak város közötti út hosszát jelölik. Feladat: az előbbi gráfban minimális Hamilton-kör keresése.

Hamilton-út: Egy P út Hamilton-út G gráfban, ha a G minden csúcsát pontosan egyszer érinti.
Hamilton-kör: Olyan Hamilton-út, amelynek a kezdő és végpontja megegyezik.

Számos gyakorlati feladat vezethető vissza TSP-re vagy annak valamely változatára, ilyenek például a szállítmányozási és járattervezési feladatok, repülőlők és azok személyzetének útiterve, vagy ha egy adott munkadarabon (pl.: chip) több (millió) lyukat kell fúrni (égetni), akkor a lyukak közti útvonalat célszerű optimalizálni. [4] [8]

1.2. Komplexitás

A Hamilton-kör megtalálása egy gráfban NP-nehéz feladat, így a TSP és valamennyi változata is szintén NP-nehéz, tehát nem létezik olyan polinomiális idejű algoritmus, amely használatával meghatározható lenne az optimális megoldás, ha $P \neq NP$. (A továbbiakban implicit feltesszük, hogy $P \neq NP$.) Egy másik megközelítés a komplexitás mérésének az, hogy heurisztikus közelítő algoritmusokkal milyen "jó" megoldást lehet elérni. Ennek egy mérőszáma az aszimptotikus hányados, ami azt fejezi ki lényegében, hogy egy heurisztikus algoritmus megoldása mennyivel tér el a feladat tényleges

optimum értékétől.
C-approximációs heurisztikus algoritmus megoldásának célfüggvény értéke legfeljebb C-szerese a feladat optimális megoldásának.
Tetszőleges polinomkorlátos TSP heurisztikához nem adható meg aszimptotikus hányados.(S. Sahni és T. Gonzalez 1976)
Általános esetben a TSP bonyolult feladat kiszámíthatósági szempontból; azonban léteznek olyan speciális osztályai amelyekre heurisztikus, sőt approximációs algoritmusokkal igen jó közelítő megoldást lehet adni és jó hír, hogy a legtöbb gyakorlatban is megjelenő probléma ezekbe az osztályokba vezethetőek vissza.
Egyelőre nem foglalkozunk a feladatok kiszámíthatóságával, mivel még nem ismerjük az egyes feladat típusok mögött álló matematikai modelleket. A következő fejezetben kerül bemutatásra a TSP feladat és fontosabb változatainak matematikai modelljei. [4] [8]

2. TSP feladat és invariánsai

2.1. TSP

Legyen i és j G két csúcsa, ekkor az alábbi egész értékű lineáris programozási (ILP) feladat adható meg: [8]

$$x_{ij} = \begin{cases} 1 & \textit{ha van (közvetlen) út } i \textit{ és } j \textit{ között} \\ 0 & \textit{egyébként} \end{cases}$$

Jelölje c_{ij} az i és j által kifeszített él súlyát, jelen esetben a két város távolságát.
A TSP feladat egy általános célfüggvénye:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \to \min$$

feltéve, hogy:

minden i csúcsot elhagyunk pontosan egyszer:

$$\sum_{j=1}^{n} x_{ij} = 1$$

minden j csúcsba bemegyünk pontosan egyszer:

$$\sum_{i=1}^{n} x_{ij} = 1$$

összes részkör kiszűrése:

$$\sum_{i \epsilon S} \sum_{j \epsilon V-S} x_{ij} \leq 1$$

$$V \epsilon [1 \ldots n]$$

2.2. CVRP

CVRP (Capacity Vehicle Route Planning), másnéven **kapacitásos gépjármű útvonal tervezési probléma**. Adott: [8]

- $G(V, E)$ súlyozott gráf
- v_0 kezdőpont (továbbiakban: depo)
- k db jármű
- c kapacitás korlát minden járműre
- d_p jelölje a p pontban szállítandó mennyisé

Keressünk k db V_0-t tartalmazó körutat hogy:

- $\forall i - re : v_i$ pontosan egy körúton van
- $\forall c_i - re : \sum_{p \epsilon c_i} d_p \leq c$

Cél: A körutak összhossza minimális legyen.

A CVRP az alábbi ILP feladattal írható fel:

$$x_{ij} = \begin{cases} 1 & \textit{ha van (közvetlen) út } i \textit{ és } j \textit{ között} \\ 0 & \textit{egyébként} \end{cases}$$

Célfüggvény:

$$\sum_{i=0}^{n} \sum_{j=0}^{n} x_{ij} D_{ij} \rightarrow min$$

minden i csúcsot elhagyunk pontosan egyszer:

$$\sum_{j=1}^{n} x_{ij} = 1$$

minden j csúcsba bemegyünk pontosan egyszer:

$$\sum_{i=1}^{n} x_{ij} = 1$$

k él megy be és k él jön ki a dopóból:

$$\sum_{j=0}^{n} x_{0j} = \sum_{i=0}^{n} x_{i0} = k$$

$$^{*}\sum_{i \epsilon S} \sum_{j \epsilon [V-S]} x_{ij} \geq r(S)$$

,ahol r(S): minimális járműszám, ami az S-beli kérések elszállításához szükséges.

Ez az ILP feladat felírás szinte egyértelműen következik a megoldandó problémából, azonban némi átalakítással könnyebben kiszámolható formára hozható feladat. Cseréljük ki, a *-gal megjelölt feltételt az alábbival:
$u_i - u_j - cx_{ij} \geq d_j - c$,ahol u_i: az i pont után a jármű által szállított mennyiség. [8]

3. TSP megoldó algoritmusok

3.1. Alkalmazásuk

A TSP feladat NP-nehéz, ezért a feladat teljes megoldásterének bejárása általában nem célravezető stratégia. A '70-es évek óta számos hatékony algoritmus látott napvilágot; klasszikus megoldásnak számítanak a mohó algoritmusok és a B&B, amelyek tárgyalása a fejezet későbbi részében következnek. Ezen a kutatási területen jellemző, hogy a már meglévő algoritmusokra számos gyorsítást találnak ki, például más tipusú megoldó eljárásokkal ötvözik a kiindulásit vagy meghatároznak egy szűkebb feladat osztályt és annak speciális tulajdonságait felhasználva érnek el javulást.

A későbbiekben bumetatott algoritmusok csoportosítása:

- Leszámlálásos
  - "Brute force"
  - (Fibonacci)
  - (Newton)
- Véletlen választásos
  - Evolúciós → genetikus algoritmus
  - Hangyakolóniás keresés
  - Tabu keresés
  - Szimulált hűtés
- Körút építő
  - Legközelebbi város hozzáadása
  - Legközelebbi város beszúrása
  - Legolcsóbb város beszúrása
  - Legtávolabbi város hozzáadása
  - Branch & Bound

A "nyers erő" módszere csak kis számú (13 db) város esetén talál optimális megoldást elfogadható időn belül. Például 13 városra kb.: 1 nap alatt számolható ki az optimális úthossz. 14 városra már nem érdemes próbálkozni.[1]

A különböző algoritmus típusok összahasonlításának nincs sok értelme, mert feladat szerkezet, de leginkább implementáció függő, hogy milyen eredményt érnek el.

3.2. Hozzárendelési feladat

Ha megnézzük a TSP matematikai modelljét úgy, hogy elhagyjuk a harmadik feltételt, akkor a hozzárendelési feladatot kapjuk. Ha a hozzárendelési feladat lehetséges megoldásainak halmaza S, a TSP-é pedig L, akkor $L \subset S$. Kézenfekvő a gondolat, hogy járjuk be az S halmazt és vizsgáljuk meg, hogy hol vesz fel a célfüggvény érték minimumot, ezt nevezzük teljes leszámolási eljárásnak, azonban ez nem célravezető megoldás, mert általában az S halmaz túl nagy.
A hozzárendelési feladat optimum értéke a hozzá tartozó TSP optimumának $99,2\%$-a(!), ezért a későbbiekben megismerkedünk több olyan algoritmussal is, amelyek egy hozzárendelési feladatból indulnak ki és annak eredményét járják be valamilyen heurisztikával vagy iteratívan javítják azt.

3.3. Legközelebbi város hozzáadása (Nearest addition)

Itt egy egyszerű mohó algoritmusról van szó, úgy működik, hogy mindig az utoljára megtalált pontból megkeresi és hozzáfűzi a hozzá legközelebbi várost. Ha már nincs több város, akkor az utolsó pontot összeköti az elsővel.

Futási idő: Naív módszerrel On^3 a lépés szám, de a távolságok tárolásával $O(n^2)$ -re lehet csökkenteni.

A megoldás a legrosszabb esetben $\frac{1}{2}\lfloor \log_2(n) \rfloor + \frac{1}{2}$ -e az optimálisnak.[8]

3.4. Legközelebbi város beszúrása (Nearest insertion)

Ez az előzőnél annyival intelligensebb módszer, hogy itt nem az utoljára vett városhoz képest veszi a legközelebb esőt, hanem az összes eddig megtalált városhoz képest keresi a legközebbi olyat, amelyik még nincs a részkörútban és azt szúrja be a legolcsóbb helyre, tehát ha a $c_{ii'}$ volt a minimális, akkor az i után i', majd a j város következik.

Futási idő: $O(n^2)$

A heurisztika eredménye legfeljebb kétszer rosszabb eredményt ad, mint az optimális.[8]

3.5. Legolcsóbb város beszúrása (Cheapest insertion)

Azt a pontot szúrja be a részkörútba, amely hatására a legkevésbé nő a részkörút összköltsége. Az eljárás műveletigénye $o(n^3)$ és asszimptotikus hányadosa 2.

Futási idő: $O(n^2 \log_2(n))$

A heurisztika eredménye legfeljebb kétszer rosszabb eredményt ad, mint az optimális, hasonlóan a legközelebbi város beszúrása algoritmushoz.[8]

3.6. Legtávolabbi város hozzáadása (Farthest addition)

A körúttól legtávolabbi pontot szúrja be minimális költséggel. Általában jobb megoldást ad az algoritmus mint az előzőek (Teszteltem, valóban így van).

Futási idő: $O(n^2)$

Legrosszabb esetben a heurisztika megoldása $2\ln(n) + 0.16$ -szorosa az optimálisnak

3.7. Korlátozás és szétválasztás (B&B)

Az egyik leggyakrabban használt eljárás a kombinatorikus optimalizálási feladatok megoldására. Az alapötlet, hogy írjuk fel a kiindulási TSP-hez tartozó hozzárendelési feladatot, ekkor a lehetséges megoldások terét fa adatszerkezettel ábrázoljuk és a fa azon ágait nem értékeljük ki, amelyek valószínűleg nem tartalmaznak optimális megoldást.
Ennek eléréséhez definiálni kell az L lehetséges megoldások terén két függvényt:

A szétválasztó függvény felelős azért, hogy a lehetséges megoldások bármely $|L'| > 1$ részhalmazához hozzárendeli L' egy valódi osztályozását.

A korlátozó függvény bármely $z(\overline{x})(\overline{x}\epsilon L')$-re meghatároz egy alsó (maximalizálás esetén felső) korlátot.

Ha a kiszámított alsó korlátnál már van jobb megoldás, akkor az adott részfát nem kell tovább kiértékelni.
Legyen φ a szétválasztó, g pedig a korlátozó függvény, ekkor a leszámolási (B&B) fát a következő eljárással építjük fel:

1. lépés gyökér := L, $r := 0$
2. lépés Határozzuk meg $g(L)$-t és rendeljük címkeként az L szögponthoz iteratívan:
3. lépés Az aktuális fa levelein határozzuk meg a címkék minimumát és válasszunk ki egy minimális?címkéjű levelet.
4. lépés Ha $L' = \{\overline{x}\}$, akkor return $\overline{x}$
egyébként következő lépés
5. lépés Bővítsük ki az aktuális fát $\varphi(L')$ elemeivel, mint L' leszármazottaival, majd az új szögpontokhoz rendre számítsuk ki a korlátokat és rendeljük az illető szögpontokhoz címkeként.
r++;
Jöjjön a 3. lépés!

Az eljárás **helyes** és **véges** időben befejeződik. A futási időt tulajdonképpen a két függvény minősége határozza meg. [8] [6]

3.8. Korlátozás és vágás (B&C)

Ha egész értékű TSP-ről van szó, tehát $x \epsilon 0, 1$, ekkor ha a B&B fa egy csúcsa által reprezentált relaxált részfeladat megoldásakor sérül az egészértékűségi feltétel, akkor vágással pontosítható az optimum alsó korlátjának értéke. Az alapötlet az, hogy a lehetséges megoldások halmazából metsző síkokkal vágjuk le azokat a darabokat, amelyek nem tartalmaznak egész értékű (koordinátájú) megoldásokat.
A **Gomory-féle metszés** (1958) volt az első olyan eljárás, amely véges időben meghatározta a szeparáló síkokat.

Lemma: A Gomory-metszés levágja az aktuális relaxált feladat $x_i \star = b$ megoldását (ha nem egész), és nem vág le egész megoldásokat.[8] [2] [3]

3.9. Paching

Ha megoldjuk a TSP feladathoz tartozó hozzárendelési feladatot, akkor egy vagy több részkörutat kapunk. Ha egy részkörutat kapunk, akkor az valójában az eredeti TSP optimális körútja, egyébként az optimális hozzárendelésnek megfelelő gráf a diszjunkt részkörutat egyesítése. Jó heurisztikának tűnik ezen részkörutak minél kisebb költségű összekapcsolása; az alábbi két eljárás ezt hivatott elvégezni.[7]

3.9.1. 2-patching

Összekapcsolás: Válasszunk ki két részkörutat: I, J. $I = \{i_1, i_2, ..., i_p, i_q, ...\}$ és $J = \{j_1, j_2, ..., j_p, j_q, ...\}$ városok. Töröljük az (i_p, i_q) és (j_p, j_q) éleket és húzzuk be az (i_p, j_q) és (j_p, i_q) éleket, ekkor az új részkörút tartalmazza az $I \bigcup J$ gráf városait a költség változás pedig:

$$d(i_p, j_q) + d(j_p, i_q) - d(i_p, i_q) - d(j_p, j_q)$$

Eljárás:(R.M.Karp)
1.lépés: Oldjuk meg a hozzárendelési feladatot, ha a hozzárendelési feladat optimális megoldása körút, akkor vége az eljárásnak, egyébként vesszük a 2.

lépést.
2.lépés: Kapcsoljuk össze 2-patching eljárással a két legkisebb elemszámú körutat. Ha az új megoldás körút, akkor visszatérünk a megoldással, egyébként következik a 2. lépés.

3.9.2. 3-patching

Általában növelhető az eljárás hatékonysága, ha egyszerre nem kettő, hanem három részkört fűzünk össze.
Összekapcsolás: Válasszunk ki három részkörutat: I, J, K $I = \{i_1, i_2, ..., i_p, i_q, ...\}$, $J = \{j_1, j_2, ..., j_p, j_q, ...\}$ és $K = \{k_1, k_2, ..., k_p, k_q, ...\}$városok. Töröljük az (i_p, i_q), (j_p, j_q) és (k_p, k_q) éleket, valamint húzzuk be az (i_p, j_q), (j_p, k_q) és (k_p, i_q) éleket, ekkor az új részkörút tartalmazza az $I \bigcup J \bigcup K$ gráf városait a költség változás pedig:

$$d(i_p, j_q) + d(j_p, k_q) + d(k_p, i_q) - d(i_p, i_q) - d(j_p, j_q) - d(k_p, k_q)$$

Eljárás:
1. lépés: Megoldjuk a hozzárendelési feladatot D-n
2. lépés: Ha kevesebb, mint 9 körút van, akkor jöjjön a 3. lépés, egyébként:
Rendezzük sorba a részkörutakat hossz szerint: $k_1, ..., k_m$
Számoljuk ki a d_{rs} 2-patching költségeket a k_r, k_{m-l+s} részkörutakra, minden $1 \leq r \leq l, 1 \leq s \leq l$ indexpárra, ahol $l = [M/2]$.
Megoldjuk a d_{ij} költségmátrixú hozzárendelési feladatot és a megoldásban kapott részkörút párokat rendre összefűzzük, úgy hogy a nagy köröket a kis körökkel párosítjuk arra törekedve, hogy az összefűzés után ne legyen túl nagy a költség növekedés.
Folytassuk az 1. lépéssel!
3. lépés: Ha 1 körút van, akkor optimumban vagyunk.
Ha 2 körút van, akkor összefűzzük őket és az az optimális megoldás.
Ha 3 $\leq$ körút van, akkor a 3 legrövidebbet összefésüljük és folytatjuk a 3. lépéssel.

4. Metrikus TSP

4.1. Feladat

Ha egy TSP feladatra teljesül az alábbi két feltétel:

- **Szimmetrikus költségmátrix**, azaz A városból B-be eljutni ugyan akkora költséggel lehet, mint B-ből A-ba.
- **Háromszög egyenlőtlenség**, azaz $d_{ij} + d_{jk} \geq d_{ik}, \forall(i, j, k)$

ez továbbra is NP-nehéz feladat, de létezik hozzá approximációs hányados.

4.2. 2-approximációs algoritmus

Jelöljük G-vel a TSP-t ábrázoló gráfot, ekkor:
1. lépés: Keressünk G-n minimális feszítőfát Prim vagy Kruskal algoritmussal.
2. lépés: Járjuk be a feszítő fát preorder eljárással; ekkor kapunk egy minden csúcsot legalább egyszer érintő körutat.
3. lépés: Minden pontnak csak az első előfordulását tartsuk meg.

Tétel: A 2-approximációs eljárás approximációs hányadosa 2.[8]

4.3. Christofides algoritmus

Az eljárás ismertetése előtt következzen két gráfelméleti fogalom, amelyekre szükség lesz ahhoz, hogy megértsük az algoritmus működését.
Minimális párosítás: G gráfban minimális párosítás azon $c_1, ..., c_n$ élek halmaza, amelyeknek nincs közös pontjuk.
Teljes párosítás: Ha minden csúcs valamelyik párosításbeli élnek a pontja.
Algoritmus:
1. lépés: Határozzuk meg a minimális feszítőfát.
2. lépés: A páratlan fokszámú csúcsok által feszített részfában keressünk minimális teljes párosítást, úgy hogy eggyel növeljük a fokszámot, ezzel kiegészítve a fát.
3. lépés: Euler-kör építés.

Minden pontnak az első megjelenését tartjuk meg.

Tétel: A Christofides algoritmus approximációs hányadosa 2/3.
[8] Futási idő: $O(n^2 log_2 n)$

5. Globális optimalizálás

5.1. Feladat környezet

Az eddigi eljárásoknál az optimumot vagy egy ahhoz közeli állapotot kerestünk és a költség az oda vezető út függvénye volt. A globális optimalizálási problémáknál a költség az állapot függvénye, tehát nem az úté.[9]
A globális optimalizálási modelje:

- lehetséges állapotok halmaza
- kezdőállapot(ok), végállapot(ok)
- lehetséges operátorok halmaza és egy átmenet függvény
- kiértékelő függvény (f), mely minden lehetséges állapothoz valós értéket rendel

5.2. Hegymászó keresés és javításai

Mohó megközelítés, az alap hegymászó algoritmus csak lokális optimumot talál meg.
Hegymászó:
1: aktuális állapot ← véletlen állapot
2: szomszéd ← aktuális állapot egy max értékű szomszédja
3: if $f(szomszd) \leq f(aktulis)$ return aktuális
4: else akutális ← szomszéd
5: goto 2

Hegymászó javításai:

Sztochasztikus hegymászó: a szomszédok közül véletlenül választ, az aktuális állapotnál jobbat, de nem feltétlenül a legjobbat. Lassabb, de ritkábban akad el, mint az egyszerű hegymászó.

Véletlen újraindított hegymászó: Ha nem találtunk célállapotot, akkor véletlen kezdőpontból indítsuk újra az algoritmsut. Fontos, hogy itt az egyes keresések között nincs információ megosztás. Bár sok függ a keresési tér strukturájától, de ez egy nagyon hatékony algoritmus.

5.3. Szimulált hűtés

Alapötlete a fémöntés techninkájának analógiáján nyugszik lényege, hogy az aktuálisnál rosszabb értékeket is elfogad egyre csökkenő valószínűséggel. Legfontosabb fogalom a *hűtési terv*, ez határozza meg, hogy milyen valószínűséggel fogadunk el egy újabb "rossz" értéket.

Szimulált hűtés
1: aktuális $\leftarrow$ véletlen állapot; t $\leftarrow$ 0
2: $t \leftarrow t + 1; T \leftarrow$ hűtési-terv(t)
3: if $(T == 0)$ return aktuális
4: szomszéd $\leftarrow$ aktuális egy véletlen szomszédja
5: $d = f($ szomszéd $) - f($ aktuális $)$
6: if $(d > 0)$ aktuális $\leftarrow$ szomszéd
7: else aktuális $\leftarrow$ szomszéd $\exp(d/T)$ valószínűséggel
8: goto 2:
Megfelelő hűtési tervvel megtalálható vele a globális optimum.[9]

Futási idő: A hűtési tervtől sok minden függ, de általában $O(n^2)$ körüli idő várható.

5.4. Nyaláb keresés

Populáció alapú keresés, azaz nem egy aktuális állapot van, hanem K db, ezt hívjuk populációnak és a teljes populáció hatással van a következő populáció kiválasztására.[9]

Nyaláb keresés

1: aktuális[] $\leftarrow$ K véletlen állapot

2: generáljuk mind a K állapot összes szomszédját

3: aktuális[] $\leftarrow$ K legjobb az összes szomszéd közül

4: if aktuális[i] célállapot valamely i-re return aktuális[i]

5: goto 2:

5.5. Genetikus algoritmus

Ez is populáció alapú keresés, de ki van bővítve néhány operátorral:

Kombináció (cross over): a populáció egyedeit kettesével keresztezzük. Itt azonban nem működik a hagyományos kereszezés, amit a String-eknél használhatunk, mert akkor a keresztezés után nem részkört kapnánk, tehát oda kell figyelni, hogy a TSP szabályt ne sértsük.

Mutáció: Néhány egyedben az egyedeken belül felcserélünk egy vagy több elemet.

Szelekció: az új populáció egyedeinek kiválasztásához fitness (jósági vagy cél-) függvényt használunk.

Hatékony genetikus algoritmus jelenleg nem ismert TSP-re. Szokás azonban keverni más algoritmussal, például a mutáció egy lokális keresés. [9]

Futási idő: Az időt meghatározza a feladat méretén kívül a populációk mérete. A kilépési feltétel lehet egy adott fitness érték elérése vagy adott számú populáció lefutása.

5.6. Tabu keresés (TS)

Mostanába nagyon felkapott és ténylegesen is hatékony meta-heurisztika. Az ötlet, hogy:

- az aktuális populáció mellett eltároljuk az eddigi legjobb csúcsokat
- készítünk egy tabu listát az utóbbi néhány aktuális csúcsból

A tabu lista célja az, hogy elkerüljük a visszalépést egy nem túl régen meglátogatott csúcsba.

Aspiráns kritérium: néha meg lehet szegni a tabu listát, például ha jobb megoldást ad, mint az eddigi legjobb

Szétterjesztés (diverzifikáció): A még fel nem derített részeket is meglátogatjuk.

Felerősítés: A jó értékek körüli környezetet alaposabban be kell járni.

Jelölt lista: Ha túl nagy az aktuális csúcsok környezete, akkor csak a k legjobbat vizsgáljuk.

Iteratívan **1:** kiválasztjuk a legjobb csúcsokat az aktuális csúcsok környezetéből (kivéve a tabu listát)
2: ha az új csúcs jobb, mint az eddigi legjobbak közül valamelyik, akkor azt lecseréljük
3: a fentiek alapján módosítjuk a tabu listát

Kilépési feltétel:

- ha a célfüggvény az eltárolt legjobb csúcsok halmazán optimális
- ha az adott populáció vagy a legjobb eltárolt csúcsok halmaza sokáig nem változik
- túllépünk a költségkorláton (pl.: időkorlát)

[8]

5.7. Hangya kolóniás keresés (ant colony system)

A természetben a hangyák táplálék keresés közben a tápláléktól a bolyhoz vissza vezető úton feromont bocsátanak ki magukból, amit más hangyák is

nagy valószínűséggel követni fognak, ha arra járnak és további feromont bocsátanak ki az úton, ezzel is növelve az úton a feromon szintet. Azonban a feromon folyamatosan párolog, amíg el nem éri a nulla szintet (érdemes lehet nem iterációnként csökkenteni az összes út feromon szintjét, mert akkor nehezen tudunk felderítést végezni és esetleg csak lokális optimumok körül mozgunk, hanem csak csak az adott iterációban létrehozott utokon csökkentsük).

Egyes hangyafajokra jellemző, hogy ha jobb minőségű táplálékot (jobb megoldást) találnak, akkor több fermont bocsátanak ki a vissza úton, ezzel növelve annak valószínűségét, hogy más hangyák (ágensek) is azon az úton induljanak el. Ha ugyan ahhoz a táplálékhoz több út is vezet, akkor a rövidebb úton gyakrabban fordulnak a hangyák, aminek következtében magasabb lesz a feromon szint az adott úton, aminek következtében még több hangya választja azt az utat és így egy idő után a másik út kiválasztódásának valószínűsége minimálisra csökken. Az egyes utakon a feromon szinteket az F Feromon-mátrixban tartjuk nyilván; f_{ij} az ij csúcsok által kifeszített út feromon szintjét mutatja.

Önmagában egy hangya, azaz az ágens igen korlátolt képességekkel bír, azonban a teljes hangya kolónia igen hatékonyan oldja meg a különböző kombinatorikus optiamlizálási feladatokat, ráadásul könnyen adoptálható az egyes feladat típusok között, így hozzárendelési feladatra, hátizsák feladatra, ütemezési és TSP feladatokra is hamar elkészültek a megoldó eljárások.

$$p_{ij} = \frac{f_{ij}^{\alpha} \cdot \frac{1}{d_{ij}}^{\beta}}{\sum_{k \epsilon L} [\frac{1}{d_{ij}}^{\beta}]}$$

A fenti képlettel az i városból a j városba menés valószínűségét számoljuk ki, ahol L a még látogatható városok halmaza, α és β pedig paraméterek, amiket a futtatások során pontosíthatunk. f_{if} és d_{ij} továbbra is rendre jelöljék a feromon szintet és a távolságot ij csúcsok között.

A **futási idő** függ a kilépési feltételtől és a futtatás során beállított paraméterektől. [5]

Hivatkozások

[1] http://www.dc.fi.udc.es/lidia/mariano/demos/tsp/lab3b.html.

[2] http://www.rpi.edu/ mitchj/papers/mitche2.pdf.

[3] V. Béla. Egész értékű programozás, typotex, 2006.

[4] T. Csendes. Optimalizálás alkalmazásai jegyzet.

[5] J. Dombi. Intelligens rendszerek, előadásjegyzet.

[6] C. Imreh. http://www.inf.u-szeged.hu/ cimreh/5.pdf.

[7] C. Imreh. http://www.inf.u-szeged.hu/ cimreh/8.pdf.

[8] C. Imreh. Utazó ügynök és járattervező algoritmusok, órai jogyzet.

[9] M. Jelasity. Mesterséges intelligencia előadás jegyzet, 2009.