Heurisztikák algoritmusok ütemezési problémákra

1. Állapottér és a megoldások kezelése

Számos nehéz ütemezési probléma esetén az exponenciális idejű optimális megoldást adó algoritmusok rendkívül nagy időigénye nem teszi lehetővé, ezen algoritmusok használatát nagyobb méretű gyakorlati problémák megoldására. Továbbá több modell esetén nem ismert olyan egyszerű közelítő algoritmus, amely véges legrosszabb eset-korláttal rendelkezik, illetve vannak olyan problémák is, amelyek esetén ismert, hogy bizonyos általában igaznak tartott komplexitáselméleti feltételek mellett nincs is konstans legrosszabb eset-korláttal rendelkező polinomiális idejű közelítő algoritmus. Ilyen esetekben jól használhatók a lokális kereső algoritmusok.

Ebben a részben röviden vázoljuk miként használhatók a lokális kereső algoritmusok ütemezési problémák megoldására. Nem tekintjük át az egész területet, csak a szomszédsági keresésen alapuló algoritmusokat tárgyaljuk. Ebben az esetben sem tekintünk konkrét ütemezési problémákat, csak néhány alapelvet mutatunk be, néhány általános módszert arra, miként reprezentálhatjuk a megoldásokat, és miként definiálhatunk megfelelő szomszédsági környezeteket. Ezen terület részletesebb tanulmányozása túlmutat a jegyzet keretein, az érdeklődő olvasó több részletet megtalál a [**?**] surveyben és az ott szereplő irodalomjegyzékben.

1.1. Reprezentáció

A lokális kereső algoritmusok használata során rendkívül fontos miként reprezentáljuk a lehetséges megoldásokat. Egyes esetekben előfordulhat, hogy a lehetséges megoldások olyan struktúrát alkotnak, amelyen nem találunk megfelelő szomszédságfogalmat, ilyen esetekben jól használhatóak absztrakt reprezentációk melyek valamilyen értelemben hozzárendelhetők a lehetséges megoldásokhoz. A reprezentációban használt objektumok jobban használható struktúrát alkothatnak, hiszen megadhatunk olyan reprezentációt, amelyben egy lehetséges megoldáshoz több objektum is tartozik.

Formálisan fogalmazva megkülönböztetjük a lehetséges megoldások $\mathcal{F}$ halmazát (vagy valamely kiválasztott részhalmazát), és a reprezentációban megadott strukturák $\mathcal{R}$ halmazát, amelyeken a lokális kereső algoritmust végrehajtjuk. A reprezentáció és a lehetséges megoldások közötti kapcsolatot egy G függvény adja meg, amely az $\mathcal{R}$ halmazból a $\mathcal{F}$ halmazba képez, és azt adja meg, hogy az adott objektum mely lehetséges megoldáshoz tartozik. Amennyiben ez a G függvény injektív, azaz minden lehetséges megoldásnak legfeljebb egy reprezentációja van, akkor teljesen mindegy, hogy a reprezentációban vagy a lehetséges megoldások halmazán dolgozik az algoritmus, hiszen a szomszédság és egyéb fogalmak könnyen transzformálhatók az egyik halmazról a másikba.

1.2. Lehetséges megoldások megőrzése

Bizonyos esetekben, amelyekben az ütemezési problémában extra feltételek adottak (pl határidők, érkezési idők) technikailag egyszerűbb olyan reprezentációt definiálni, amely olyan megoldásokhoz is rendel objektumot, amely nem elégiti ki ezeket az extra feltételeket, de jól definiálhatók a lokális kereső algoritmushoz szükséges fogalmak. Ebben az esetben azokat az objektunokat, amnelyekhez nem tartozik lehetséges megoldás nemlehetséges objektumoknak hívjuk. Hasonló eset, amikor előre tudjuk, hogy van optimális megoldás, amely kielégit bizonyos tulajdonságokat, és szeretnénk a keresést a tulajdonságokat kielégítő megoldásokra megszoritani. Ezekben az esetekben kezelnúnk kell az algoritmus során a nemlehetséges objektumokat.

- *Kizárás* Ebben az esetben nem foglalkozunk a nemlehetséges objektumokkal. Ez vagy úgy lehetséges, hogy csak olyan objektumokat generálunk, amelyek lehetséges megoldásokat eredményeznek vagy az objektumok generálása után töröljük azokat, amelyek nem tartoznak lehetséges megoldáshoz.
- *Büntetőfüggvény* Ebben az esetben figyelembe vesszük a nemlehetséges objektumokat is, de a célfüggvény értékét nőveljük egy büntető költséggel.
- *Javítás* Ebben az esetben azokat az objektumokat, amelyekhez nem tartoznak lehetséges megoldások, kijavítjuk, olyan lehetséges megoldásokat leiró objektumokat keresünk, amelyek valamilyen értelemben hasonlóak a kapott nemlehetséges objektumhoz. Ezen megoldás egy modósított változata, amelyben az eljárás során megörizzük a nemlehetséges objektumot, de a javított változat célfüggvényértékét rendeljúk hozzá.

2. Szomszédsági keresés

A lokálisan kereső algoritmusok jelentős része szomszédsági keresésen alapul. A szomszédsági kereső algoritmusok alapötlete, hogy a lehetséges megoldások terén (vagy a megoldások terének egy reprezentációján) egy szomszédsági struktúrát alakítsunk ki. Szomszédságon egy függvényt értünk, amely a reprezentációhoz tartozó objektumokon értelmezett, és minden objektumhoz az objektumoknak egy részhalmazát rendeli hozzá. Az adott szomszédsági struktúra mellett egy szomszédsági kereső algoritmus a következő alapelven működik. Az algoritmus egy kezdeti objektumot (lehetséges megoldást) választ, ez lesz az aktuális objektum. Ezt követően iteratív módon mindaddig, amíg egy megállási feltételt el nem ér, az aktuális objektum szomszédságából választ egy új objektumot és az lesz az aktuális objektum. Az eljárás során végig fel van jegyezve az eddig ismert legjobb megoldás, amelyet ha jobb megoldást találunk felülírunk.

A megállási feltételtől, és attól függően a szomszédságból, mely objektumot választjuk számos változat létezik. Külön megemlítenünk két változatot.

Az elsőt iteratív javító algoritmusnak nevezzük. Ezen algoritmus az aktuális objektumot mindig olyan objektummal cseréli ki a környezetből, amelynek

kisebb a költsége. Amennyiben nem talál ilyen objektumot, akkor az algoritmus véget ér. Ha mindig a legjobb megoldást választja a környezetből, akkor az algoritmust hegymászó algoritmusnak nevezzük. Egyszerűen látható, hogy az iteratív javító algoritmus csak azon szomszédság struktúrák esetén határozza meg az optimális megoldást, amelyekben minden *lokális optimum* (azon objektumok, amelyek célfüggvényértéke optimális a szomszédságukban) egyben globális optimum is. Amennyiben a szomszédsági struktúra nem teljesíti ezt a feltételt, és előfordulnak a globális optimumnál sokkal rosszabb célfüggvényértékkel rendelkező lokális optimumok is, gyakran használnak olyan szomszédsági kereső algoritmusokat, amelyek bizonyos valószínűséggel egy nagyobb költségű objektumot is választhat következő aktuális objektumnak. Ez esetben ügyelni kell arra, hogy ne lépjünk vissza a már elhagyott lokális optimumra. Ennek a kezelésére fejlesztették ki a tabu keresés algoritmusát.

Az alapgondolat az, hogy az algoritmus karbantart egy tabulistát is, amely azokat a megoldásokat írja le, ahova nem akarunk lépni. Ez általában nem megoldások egy listája, hanem tiltott lépéseké (pl. az alább ismertetett beszúrási szomszédságnál azokat a munkákat tartalmazza, amiket nem lehet mozgatni). Ezt a listát minden iterációban felülírjuk, az aktuális lépés visszalépésének tiltásával bővítjük, az elavult tiltásokat töröljük. Az aktuális iterációban pedig nem szabad tabu listán szereplő lépést megtenni, ezzel biztosítjuk, hogy nem lépünk vissza a megelőző megoldásra.

Általában a tabu kereső algoritmus a legjobb megoldást választja a környezetből, ha az nem az aktuális megoldás és nincs tiltva a tabulista által. (De vannak olyan változatok is,a melyek bizonyos valószínűséggel ilyen esetekben is választanak más megoldást.) Amennyiben a legjobb megoldás az aktuális megoldás - azaz lokális optimumban vagyunk - vagy a legjobb megoldást tiltja a tabulista, akkor más kevésbé jó megoldást választ az algoritmus. Érdemes még megemlíteni pár ötletet, amit tabu keresésnél használni szoktak.

A felerősítés elve arra szolgál, hogy néhány jó megoldás környékét alaposabban körbejárjuk, a szétszóródás elve pedig arra, hogy a megoldástérnek ne legyen nagy egyáltalán nem meglátogatott részei. Ezt a két elvet általában kétfázisú algoritmusokkal szoktak megvalósítani. Első fázisként több különböző pontból indított gyors (egyszerű szomszédság, kevés iteráció) tabu keresést hajtunk végre, ez a szétszóródás elvének felel meg. Majd a legjobb megoldásokból egy részletesebb kereső algoritmust indítunk. Érdemes még megemlíteni az aspiráns kritérium elvét, ami azt jelenti, hogy ha van egy az eddig ismert legjobb megoldásnál jobb elem a szomszédságban, de azt tiltja a tabu lista, akkor a tiltást felülírjuk és abba a megoldásba lépünk. Továbbá gyakran használt fogalom a jelöltlista, amit nagy méretű környezetek esetén használunk. Jelölt listák esetén mindig a környezetnek csak egy részét vizsgáljuk (pl többdimenziós változtatási lehetőségek esetén csak néhány dimenziót).

Mint a fentiekből kiderül a szomszédsági kereső algoritmusok esetén a legfontosabb lépés a jó szomszédsági struktúra kiválasztása. Az, hogy mely szomszédsági struktúrák jól használhatók több tényezőtől is függ.

Fontos lehet a szomszédságok mérete. Amennyiben ez a méret nagyon nagy,

nehéz megvizsgálni az aktuális objektum szomszédságát. Másrészt kicsi szomszédságok esetén könnyen lehetnek olyan lokális optimumok, amelyek az optimálisnál sokkal rosszabb megoldást eredményeznek.

Egy másik fontos kérdés miként lehet a célfüggvény értékét egy adott szomszédságon belül hatékonyan kiszámolni. Jelentősen meggyorsíthatja az algoritmust olyan szomszédságok megadása, amelyeken belül az aktuális érték felhasználásával könnyen kiszámítható az objektumokhoz tartozó célfüggvényérték.

Végül rendkívül fontos tényező, hogy olyan szomszédságokat igyekezzünk definiálni (főleg iteratív javító algoritmusok esetén), amelyekben nincsennek nagyon rossz célfüggvényértékkel rendelkező lokális optimumok.

2.1. Sorbaállítási problémák

Sok ütemezési probléma esetén az ütemezést egyértelműen meghatározza, hogy milyen sorrendben ütemezzük a munkákat. A legtöbb modell, amelyben csak egy gépet tekintünk ilyen.

2.1.1. Reprezentáció

Az ilyen sorbaállítási problémáknál a legtermészetesebb reprezentáció a munkák (azaz az $1, \ldots, n$ számok egy permutációja), ezt a reprezentációt *természetes reprezentációnak* hívjuk. Másrészt az ilyen típusú problémák esetén más reprezentációk is hasznosak lehetnek, az alábbiakban röviden bemutatunk néhány egyéb reprezentációt.

Amennyiben a munkákhoz precedencia feltételek is tartoznak permutációk többségéhez nem tartozik lehetséges megoldás. Ebben az esetben használhatjuk a fentiekben említett lehetséges megoldást megörző technikákat.

Ebben az esetben hasznos lehet a *prioritás reprezentáció*, amely prioritás szerint állítja sorba a munkákat. Egy munka a prioritási sorrendben mindig a precedencia gráfban előtte levő munkák után következik (a prioritás szerinti teljes rendezés a precedencia feltételek által adott parciális rendezés kiterjesztése). Tulajdonképpen ez a reprezentáció a természetes reprezentáció egy megszorítása, mivel a permutációknak csak bizonyos részhalmazait engedjük meg. Jól használható ez a reprezentáció abban az esetben is, amennyiben a munkákhoz tartozik érkezési idő vagy határidő.

érdemes még megemlíteni a rendezett pár reprezentációt. Ebben a reprezentációban minden párjához a munkáknak tartozik egy bit, amely az (A, B) pár esetén 1 ha A megelőzi B-t, és 0 különben.

2.1.2. Szomszédság

Elsőként tekintsük a természetes reprezentáció esetét. Ezen reprezentáció mellett az alábbi négy alapvető szomszédságot definiálhatunk. Példaként a 8 elemből álló $(1, 2, 3, 4, 5, 6, 7, 8)$ sorrendet használjuk.

- *Transzpoziciós szomszédság:* Ezen szomszédságfogalom mellett egy sorrend szomszédjai azok a sorrendek, amelyeket az eredeti sorrendből két

egymás melletti elem felcserélésével (ezt aműveletet hívjuk transzpozíciónak) kapunk. Például $(1, 2, 4, 3, 5, 6, 7, 8)$ a kiindulási sorrend egy szomszédja.

- *Beillesztési szomszédság:* Ezen szomszédságfogalom mellett egy sorrend szomszédjai azok a sorrendek, amelyeket az eredeti sorrendből úgy kapunk, hogy egy elemet kiveszünk és a sorrendben egy másik helyre illesztünk be. Például $(1, 2, 4, 5, 6, 7, 3, 8)$ a kiindulási sorrend egy szomszédja.

- *Cserélési szomszédság:* Ezen szomszédságfogalom mellett egy sorrend szomszédjai azok a sorrendek, amelyeket az eredeti sorrendből két elem cseréjével kapunk (a különbség a transzpoziciós szomszédsággal, hogy itt nem kótjük ki az elemek egymás mellettiségét). Például $(1, 2, 3, 8, 5, 6, 7, 4)$ a kiindulási sorrend egy szomszédja.

- *Blokk beillesztési szomszédság:* Ezen szomszédságfogalom mellett egy sorrend szomszédjai azok a sorrendek, amelyeket az eredeti sorrendből úgy kapunk, hogy egymás melletti elemek egy sorozatát kiveszünk a sorrendből és egy másik helyre illesztúnk be. Például $(1, 2, 5, 6, 3, 4, 7, 8)$ a kiindulási sorrend egy szomszédja.

Nem vizsgáljuk részletesen az egyes szomszédságfogalmak tulajdonságait. Pusztán néhány észrevételt teszünk a beillesztési szomszédsággal kapcsolatban a többi szomszédságfogalomnak is vannak hasonló tulajdonságai.

Elsőként tekintsük a szomszédságok elemszámát. Egyszerűen látható, hogy ebben a modellben minden elemnek $(n-1)^2$ szomszédja van. Valóban n féleképpen választhatjuk ki az elemet, amelyet kiveszünk a sorból, és $n-1$ különbözoző helyre illeszthetjük be. Ez $n(n-1)$ szomszédot adna, viszont minden i-re ugynazt a sorrendet redményezi, ha az i-edik elemet illesztjük az $i+1$-edik mógé, mint amikor az $i+1$-ediket az i-edik elé, így a fentiekben $n-1$ szomszédot kétszer számoltunk, azaz a szomszédok száma $n(n-1)$.

Szintén érdemes meggondolnunk miként számítható ki az eredeti sorrendhez tartozó célfüggvényértékből, a szomszédok célfüggvényértéke. Tekintsük az $1||\sum T_j$ problémát. Ekkor $j < k$ esetén a i edik munka beillesztés a k-adik helyre, csak azon munkákra változtatja meg a T_j értéket, amelyek az i-edik és k-adik munka között helyezkedtek el. Az i-edik munka befejezési ideje a $\sum_{j=i+1}^{k} p_j$ értékkel nőtt, a többi $j = i+1, \ldots, k$ indexre T_l a $\min\{p_i, T_l\}$ értékkel csökken. ezen észrevételek alapján könnyen kiszámítható egy elem szomszédjaira a céfüggvényérték.

2.2. Hozzárendelési típusú problémák

Számos ütemezési probléma úgy fogható fel, hogy a munkáknak a gépekhez egy optimális hozzárendelését kell megtalálnunk. Ilyen feladat az összes olyan párhuzamos gépekre vonatkozó ütemezési probléma, amelyben a célfüggvény nem függ az egyes gépekhez rendelt munkák sorrendjétől (vagy egyszerűen meghatározható az adott hozzárendelés mellett optimális sorrend).

2.2.1. Reprezentáció

Az olyan problémák esetén, amelyekben n munkát ohajtunk m géphez hozzárendelni, a *természetes reprezentáció*, m darab halmaz amelyben az egyes halmazok az egyes gépekhez rendelt munkákat tartalmazzák.

Ebben az esetben is használható a *prioritás reprezentáció*. A prioritási sorrend alapján úgy kapjuk meg az egyes gépekhez rendelt munkákat, hogy a prioritási sorrend alapján a munkákat az előző fejezetben ismertetett Lista algoritmussal ütemezzük.

2.2.2. Szomszédság

Tekintsük a természetes reprezentáció esetét. Ezen reprezentáció mellett az alábbi három alapvető szomszédságot definiálhatjuk.

- *Gépváltási szomszédság* Ezen szomszédság mellett egy hozzárendelés szomszédjait úgy kapjuk meg, hogy valamely elemet eltávolítunk az egyik halmazból (egyik gépről) és egy másik halmazba (másik gépre) helyezzük át.
- *Cserélési szomszédság* Ezen szomszédság mellett egy hozzárendelés szomszédjait úgy kapjuk, hogy kicserélünk két különböző halmazban levő elemet.
- *2-Gépváltási szomszédság* Ezen szomszédság mellett egy hozzárendelés szomszédjait úgy kapjuk meg, hogy egy vagy két munkát eltávolítunk az egyik halmazból és egy másik halmazba vagy halmazokba helyezzük át.

2.3. Kombinált hozzárendelési és sorbaállítási problémák

Számos ütemezési problémánál hozzárendelési és sorbaállítási feladatot is meg kell oldanunk. Ebbe az osztályba tartoznak azok a párhuzamos gépekre vonatkozó problémák, amelyek során az egyes gépekhez rendelt munkák sorrendjét is figyelembe kell vennünk.

2.3.1. Reprezentáció

A hozzárendelési problémák esetén használt reprezentáció könnyen módosítható erre az esetre is. A *természetes reprezentáció* m darab rendezett lista, mindegyik az egyes gépekhez rendelt munkák sorrendjét tartja számon. A hozzárendelési esetben használt *prioritás reprezentáció* változtatás nélkül alkalmazható ebben az esetben is, hiszen az a reprezentáció kezeli a munkák sorrendjét.

2.3.2. Szomszédság

A sorbaállítási problémák esetén használt szomszédsági fogalmak kiterjeszthetők erre az esetre is. A beillesztés ebben az esetben azt jelenti, hogy egy elemet (vagy egy blokkot) kiveszünk a helyéről és beillesztjük ugyanazon a listán egy maśik helyre vagy egy másik listába. A csere ebben az esetben két tetszőleges elem kicserélését jelenti.

3. Genetikus algoritmus

A genetikus algoritmus esetén nem minden lépésben egy megoldást tartunk számon a megoldástérből, hanem minden iterációban egy a megoldástér elemeiből álló populációnk van. Egy konkrét genetikus algoritmus létrehozásához az alábbi részleteket kell specifikálnunk

- A kezdeti populáció létrehozása
- Az új elemek képzéséhez a szülők kiválasztása, továbbá a túlélő elemek kiválasztása
- A szülők keresztezésével (rekombinációjával) új egyedek konstruálása
- Mutációk végrehajtása
- Új populáció megkonstruálása
- Megállási feltételek megadása

A továbbiakban bemutatuunk néhány ötletet, amit az ütemezési problémáknál használni szoktak. A reprezentáció. amit a megoldástérnél használunk az alábbiakban permutációk tere lesz.

3.1. A kezdeti populáció létrehozása

A kezdeti populációt legegyszerűbb véletlenszerűen generálni. A populáció mérete a probléma természetétől függ, de leggyakrabban néhány száz vagy néhány ezer egyedből áll. Hagyományosan az egyedek a keresési téren egyenletesen oszlanak el, viszont egyes esetekben olyan részeken több egyedet generálnak, ahol sejthető az optimum.

3.2. Az új elemek képzéséhez a szülők kiválasztása, továbbá a túlélő elemek kiválasztása

Az új elemeket általában egy fitness függvény alapján választjuk. Ez a fitness függvény gyakran megegyezik a probléma célfüggvényével, de előfordulhat, hogy más szempontokat is figyelembe vesznek. A cél az, hogy azokat az egydeket válasszuk ki szülőnek, akiknek jók a fitness értékei, ezt kétféleképpen érhetjük el. Egyrészta legjobb fitness értékű egyedeket determinisztikusan kiválaszthatjuk a szülők közé. A másik megközelítés véletlen döntéseken alapul, a fitness értékek alapján definiált eloszlás szerint generálunk véletlenül szülőket. A genetikus algoritmusok gyakran mindkét megközelítést használják, a szülők egy részét determinisztikusan, egy másik részét véletlen eloszlás alapján választják.

3.3. A szülők keresztezésével (rekombinációjával) új egyedek konstruálása

Általában genetikus algoritmusoknál a megoldástér sztringekből áll, és ezekre jól hazsnálhatóak az egypontos illetve kétpontos kereszteződések, amelyek esetén az utódnak megfelelő sztring egy rész a az egyik a maradék része a másik szülőből jön. Ezt permutációk esetén nem tudjuk közvetlenül használni, mivel az így keletkező sztringekben lehetnek ismétlődések. Így bonyolultabb megoldásokra lesz szükség. Az alábbiakban bemutatunk két példát, miként lehet két permutációból utódokat képezni.

Sorrend alapú kereszteződés Ebben az esetben az útod az egyik szülő közepéből átvesz egy részpermutációt. Majd a maradék helyeket a permutációból hiányzó elemekkel elölről hátrafelé a másik szülőben szereplő sorrendjük alapján töltjük fel. Például ha az egyik szülő 1234567 a másik pedig 2456371, akkor amennyiben az első szülőből a középső 3 elem 345 öröklődik az utód közepére, akkor az utód 2634571 lesz, mivel a hiányzó $1, 2, 6, 7$ elemek a másik szülőben a $2, 6, 7, 1$ sorrendben szerepeltek.

Parciális leképezés kereszteződés Itt is választunk egy részpermutációt az egyik szülőből, a többi helyre pedig amásik szülőből generáljuk az elemeket. Az alapötlet, hogy minden helyre azt az elemet helyezzük, ami a másik szülőben ott van. Viszont előfordulhat, hogy ez az elem már szerepel a konstruált permutációban, ilyenkor arról a helyről probáljuk meg az új elemet kiválasztani, ahol az az elem volt, aki ezt a helyet elfoglalta. Ezt a lépést addig folytatjuk, amíg olyan elemet nem találunk, amely még nem szerepel a permutációban. Például, ha az egyik szülő 1234567, a másik pedig 7325614, akkor a következőképpen járunk el. A középső 345 részt megtartjuk az első szülőből. Ezt követően az első helyre beírjuk a másik szülőből a 7-et, a második helyre be szeretnénk írni a 3-at, de az már szerepel. Így megnézzük, hogy ki van a második szülőben azon a helyen ahol az új elemben a 3, ez a 2, ami még nem szerepel így beírjuk az utódba. A hatodik helyre az 1-et beírjuk, a hetedik helyre a 4-et szeretnénk beírni. De az már szerepel, így megnézzük, hogy ki van a második szülőben azon a helyen ahol az új elemben a 4, ez az 5, ami szintén szerepel így megnézzük, hogy ki van a második szülőben azon a helyen ahol az új elemben az 5. Ez a 6, ami nem szerepel így beírjuk, és a kapott utód a 7234516.

3.4. Mutációk végrehajtása

Az új egyedek egy részén, vagy mindegyik elemen bizonyos valószínűséggel mutációt hajtunk végre. A mutáció tulajdonképpen felfogható úgy is, hogy az adott elem környezetéből választunk egy véletlen elemet. Ilyen interpretáció mellett bármelyik a lokális keresésnél ismertetett környezet használható. Fontos megemlíteni a hibrid algoritmusokat, ahol a mutáció egy lokális kereső algoritmus végrehajtása az adott pontból.

3.5. Új populáció megkonstruálása

Az új populációt többnyire az új egyedek alkotják. Másrészt itt érdemes megemlíteni az elitizmus ötletét, amelynek használata esetén az előző populáció legjobb egyedei túlélnek és az új populációba is bekerülnek. Ha ezt az elvet hazsnáljuk, akkor az eljárás során nem kell feljegyeznünk az aktuálisan ismert legjobb megoldást, mivel az mindig ott lesz az aktuális populációban.

3.6. Megállási feltételek megadása

A genetikus algoritmusok rendszerint addig futnak, amíg egy leállási feltétel nem teljesül. Gyakori leállási feltételek a következők:

- Adott generációszám elérése.
- Ha a legjobb egyed fitness értéke már nem javul elegendő mértékben egy-egy iterációval.
- Futási idő limit.

4. Hangya kolónia keresés

A hangya kolónia algoritmus a természetből vette az alapötletét. A hangyák táplálék keresési technikáján alapul. Az alapötlet az, hogy több ágens vagy más néven hangya keres a megoldástérben, és a keresés során feromonértékeket hagynak szét. Minden iteráció két részből áll. Első lépésként a hangyák a meglevő feromonértékek illetve lokális információk alapján többnyire véletlen döntések használatával egy - egy megoldást építenek fel. Ezt követően a megoldások minősége alapján felülírják a feromonértékeket, és átlépünk az új iterációra. Az algoritmus eredményképpen a legjobb megtekintett megoldást adja vissza. Mivel a jobb megoldások környékén több feromonérték jelentkezik, ezért a hangyák ezeknek a megoldásoknak a környékét fogják egyre jobban bejárni. Az alábbiakban bemutatjuk néhány ötleten keresztül miként használhatóak ilyen algoritmusok ütemezési problémák megoldására.

1. Fázis: Megoldások építése A megoldásokat a hangyák általában lépésenként építik fel, itt csak olyan példákat nézünk, ahol a megoldás egy permutáció. Két alapvető technika van a megoldások építésére. Az első módszert olyan esetekben szokták használni, amelyekben a költségben van jelentősége annak, hogy melyik munkák következnek egymás után. Erre a legjobb példa az az egy gépes ütemezési modell, ahol minden munkapár esetén adott egy átállási idő, és amely a maximális befejezési idő minimalizálása esetén ekvivalens az utazó ügynök problémával. Ekkor a megoldás építőkockái az (i, j) élpárok, amelyek azt a döntést jelölik, hogy az i-dik munka utána j-edik jön. Amikor a hangya egy i munkát végez el, akkor egy $D(i, j)$ szerinti valószínűséggel választja következőnek a j munkát, ahol a $D(i, j)$ függvény monoton nő az (j, j) döntés $F(i, j)$ feromonértéke szerint, és monoton csökken az (i, j) döntés közvetlen költsége (átállási idő) szerint. Egy másik lehetőség az, hogy az (i,j) döntések annak

felelnek meg, hogy az i-dik helyen ütemezett munka a j munka lesz. Ekkor a még nem ütemezett munkák közül választunk véletlenül a $D(i,j)$ kiértékelési függvény szerint, ami ismét monoton nő a döntés feromonértékében és csökken a közvetlen vagy egy becsült költségben.

2. Fázis: Feromonértékek felülírása Minden döntésre a feromonérték elveszíti egy részét (ezt hívjuk párolgásnak). Majd minden olyan hangyától, amely használta a megoldásában a döntést kap valamennyi feromont, ami függ a hangya által kapott megoldás értékéből. Tehát ha $H(i,j)$-vel jelöljük azon hangyák halmazát, akik használják az (i,j) döntést, továbbá C_h-val egy hangyára az általa kapott megoldás költségét, akkor az $F(i,j)$ feromonértékek a következőképpen változnak:

$$F(i,j) = \gamma \cdot F(i,j) + \sum_{h \in H(i,j)} K(C_h).$$

A fenti képletben γ az eljárás egy paramétere a feromon megmaradási együtthatója, K pedig egy monoton függvény.