Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Informatikai Intézet
Alkalmazott Informatikai Intézeti
Tanszék

ET Erőforrás tervezés Resource Planning

2019/20 2. félév

11. rész

Dr. Kulcsár Gyula
egyetemi docens

Kereső algoritmusok alkalmazása erőforrás tervezési feladatok megoldására

Tartalom

1. Heurisztikus megoldási módszerek
2. A lokális szomszédsági keresés alapjai
3. Szimulált hűtés alkalmazása
4. Tabu keresés alkalmása

Heurisztikus megoldási módszerek

Bevezetés

- Az ütemezési feladatok többsége ***nagyon nehezen kezelhető***.
- A gyakorlatban a nagyszámú bemenő adat miatt az ***egzakt megoldási módszerek*** ritkán alkalmazhatók.
- Az esetek döntő többségében valamilyen ***heurisztikus megoldási módszer***t alkalmaznak.
- Az egyszerű feladatokra ismert megoldási módszerek a nagy feladatokban ***építőelemek***ként használhatók fel.
- Adott problémára optimális megoldást adó módszerek más ***hasonló de bonyolultabb*** feladatok esetében heurisztikus módszerként alkalmazhatók (pl. EDD, SPT, stb. szabályok).

Heurisztikus megoldási módszerek

- ***Felépítő jellegű heurisztikus algoritmusok***
  - Egy üres ütemtervből indulnak ki. Valamilyen előzetesen definiált szabály alapján kiválasztanak egy munkát és azt valamilyen stratégia alapján ütemezik be. Az ütemterv fokozatosan alakul ki.
- ***Iteratív javítás elvén működő algoritmusok***
  - Egy megengedett ütemtervből indulnak ki. Ez lehet véletlenszerűen előállított megoldás is. Apró módosításokkal megpróbálják javítani az ütemtervet a kritériumok figyelembe vételével.
- Az alap módszerek gyakran kombináltan is használatosak.

Az előadás célja

- A szakirodalomban nagyon sokféle keresési módszer fellelhető.
- Ezek részletes ismertetése nem tartozik az előadás céljai közé.
- Most a lokális szomszédsági kereső algoritmusok alkalmazásának koncepciója kerül bemutatásra példákon keresztül.

A lokális szomszédsági keresés alapjai

A keresés alapja

- A megoldandó problémát (feladatot) egy ***diszkrét optimalizálási feladat***nak tekintjük.
  - Adott egy véges S halmaz és
  - az azon értelmezett f függvény: $f:S \rightarrow R$.
  - Keressük azt az $s^* \in S$ megoldást, amelyhez a legkisebb függvényérték tartozik: $f(s^*) \leq f(s) \quad \forall s \in S$.

A lokális keresés alapelve

- 0. Lépés: Inicializáljuk a számlálót nullával: k=0. Készítsünk egy s_0 kiindulási megoldást. Legyen $s_k = s_0$ és $s^* = s_0$;
- 1. lépés: Válasszunk egy s szomszédos megoldást az s_k szomszédságából: $s \in N(s_k)$.
- 2. lépés: Vizsgáljuk meg az elfogadási kritériumot, ha az teljesül, akkor a kiválasztott megoldást megtartjuk $s_{k+1} = s$, egyébként az s_k megoldást visszük tovább: $s_{k+1} = s_k$.
- 3. lépés: Hasonlítsuk össze az aktuális megoldást az eddigi legjobb megoldással. Ha jobbat találtunk $f(s) < f(s^*)$ akkor azt jegyezzük meg: $s^* = s$.
- 4. lépés: Vegyük a következő iterációt: k = k+1. Vizsgáljuk meg a leállási feltételt, ha az teljesül, akkor készen vagyunk, egyébként folytassuk az 1. lépéssel.

Az elv alkalmazása

Ahhoz, hogy a lokális keresés elvét adott feladat (pl. ütemezési probléma) esetében alkalmazni tudjuk, a következő döntéseket kell meghozni:

- Hogyan definiáljuk a megoldás (pl.: ütemterv) reprezentációját?
- Hogyan értelmezzük a szomszédságot?
- Milyen módszert használjunk a jelölt kiválasztására?
- Mi legyen az elfogadás/elvetés kritériuma?

Egy megoldás reprezentálása

- A megoldás reprezentálása (s) függ a megoldandó feladat típusától.
- Példák:
  - Egyerőforrásos ütemezési feladatok és előzés nélküli egyutas ütemezési feladatok: az s egy vektor.
  - Párhuzamos erőforrásos, előzéses Flow Shop és Job Shop ütemezési feladatok: az S annyi vektorból áll, amennyi erőforrás szerepel a modellben.
  - stb. … (korábbi előadások).

Szomszédság

- Egy adott s_k megoldás szomszédságát értelmezhetjük úgy, hogy definiálunk egy N_x szomszédsági (módosító) operátort és azt alkalmazzuk a kiindulási (bázis) megoldáson: $s = N_x(s_k)$
- A definiált operátorral egy lépésben előállítható megoldások halmaza alkotja a szomszédságot: $N(s_k) = \{ s \mid s = N_x(s_k) \}$.

Szomszédság (1. példa)

Az egygépes megszakítás nélküli ütemezési problémák és az egyutas előzésnélküli ütemezési problémák esetében egy megoldás egyértelműen leírható ***egy vektor***ral, amelynek elemszáma megegyezik az ütemezendő munkát számával (n).

Példák szomszédsági operátorokra:

- N_1: ***két szomszédos munkát felcserél*** az ütemtervben.
  s_k = { 1, 2, 3 } $N(s_k)$ = { { 2, 1, 3 }; { 1, 3, 2 } }
  Az N1 operátor alkalmazásával n−1 számú különböző szomszédos megoldás készíthető.
- N_2: ***kiemel*** egy munkát az ütemtervből és egy másik pozícióba ***illeszti be***.
  s_k = { 1, 2, 3 } $N(s_k)$ = { { 2, 1, 3 }; { 2, 3, 1 }; { 1, 3, 2 }; { 3, 1, 2 } }
  Az N2 operátor esetében n(n−1) számú szomszédos megoldás állítható elő. Azonban ezek között vannak ismétlődések is, így a különböző megoldások száma ennél kevesebb: $(n-1)^2$.

Szomszédság (2. példa)

- Vizsgáljuk most a jóval bonyolultabb ***Job Shop*** feladatot C_{max} célfüggvény esetén.
- Ebben a feladatban egy megoldás a gépek számával (m) azonos számú vektorral írható le.

Szomszédság (2. példa folyt.)

Példa szomszédsági operátorra:

- N_1: a kritikus útvonal mentén valamely gép két szomszédos operációját felcseréli.
  A ***kritikus útvonal*** operációk sorozatából áll:
  - Az első a referencia (nulla) időpontban kezdődik.
  - A következő indítási időpontja megegyezik az előző befejezési időpontjával (és így tovább …)
  - A kritikus útvonal utolsó operációjának befejezési időpontja az ütemterv C_{max} értékével egyezik meg.

Szomszédság (2. példa folyt.)

Példa szomszédsági operátorra:

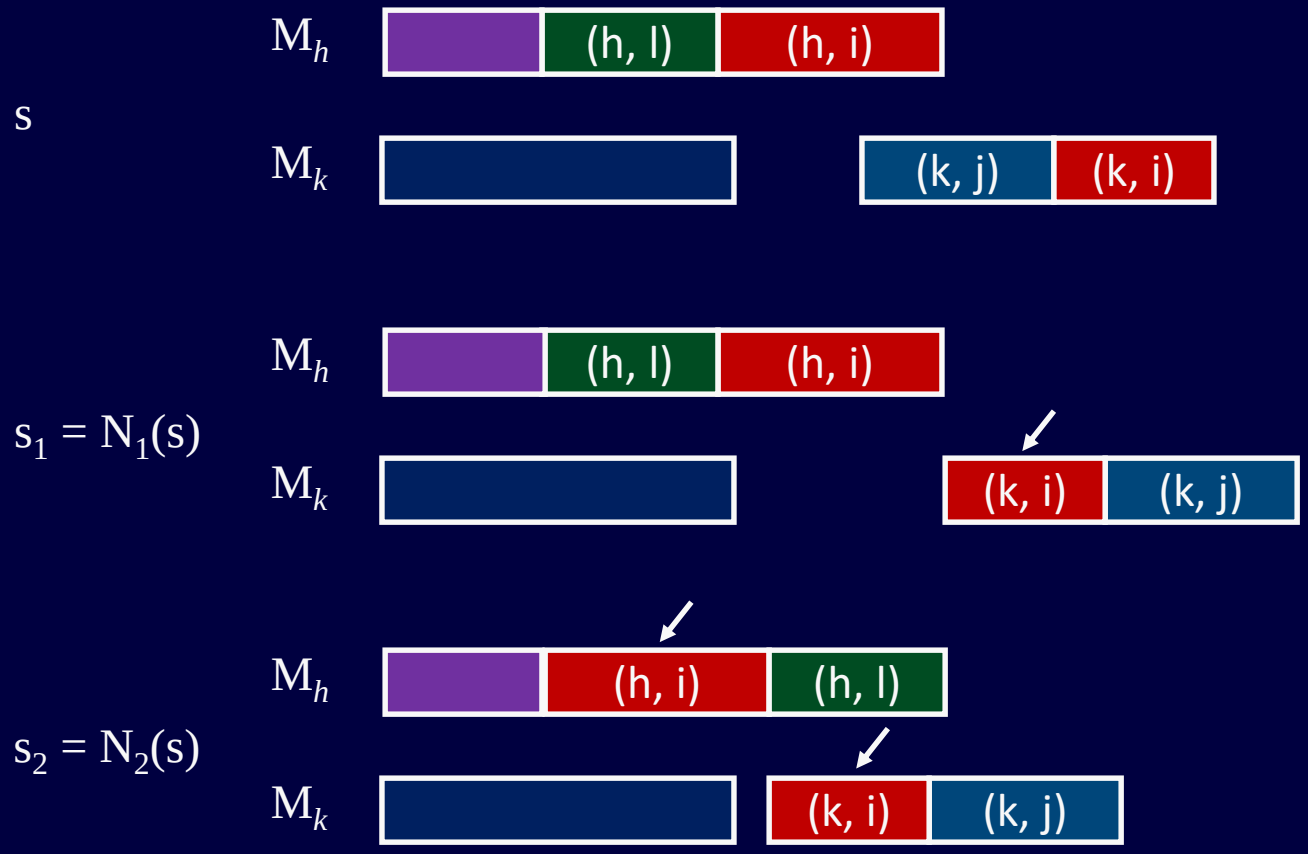
- N_2 (One Step Look Back, párcsere visszatekintéssel):
  a kritikus útvonal mentén valamely gép két szomszédos operációját felcseréli, majd az előrébb mozdított operációt tartalmazó munka eggyel korábbi operációját is felcseréli az előtte lévő operációval.

Szomszédság (2. példa folyt.)

N_2 részletesen:

- Jelölje (k,j) az M_k gépen a J_j munka operációját.
- Legyen (k,j) és (k,i) szomszédos operációk a kritikus útvonal mentén.
- Cseréljük fel a (k,j) és a (k,i) operációkat.
- Legyen (h,l) az az operáció, amely az M_h gépen közvetlenül a (k,i) operációt tartalmazó munka előtt helyezkedik el.
- Cseréljük fel a (h,l) és (h,i) operációkat.

Szomszédság (2. példa folyt.)



Kiválasztási stratégia

- A kiválasztási stratégia azt határozza meg, hogy a bázis megoldás szomszédságából ***melyik megoldást válasszuk ki*** a továbblépéshez.
- A nagyméretű feladatokban a szomszédság óriási halmazt eredményez. Ilyenkor nem vizsgáljuk meg a teljes szomszédságot, hanem valamilyen ***mintavételezés***sel a szomszédság egy szűkített részhalmazát vizsgájuk.

Kiválasztási stratégia (példák)

Sokféle mintavételezési módszer létezik.

Két tipikus példa:

- ***Véletlenszerű választás***: a bázis megoldásból véletlenszerűen állítunk elő szomszédos megoldásokat és azok közül választjuk ki a legjobbat.
- A szűkített szomszédság generálását úgy végezzük, hogy valamilyen szisztémával először azokat a szomszédos megoldásokat vizsgáljuk, amelyek a ***célfüggvény szempontjából*** várhatóan a legnagyobb javulást eredményezik.
  - Például a maximális késés minimalizálása során a módosító operátor a legnagyobb késéssel rendelkező munkákat próbálja először módosítani.

Elfogadási kritérium

- Az elfogadási kritérium azt szabályozza, hogy a szomszédságból adott lépésben kiválasztott megoldást ***elfogadjuk vagy elvetjük***.
- Ha elfogadjuk, akkor a következő iterációban abból generálunk szomszédos megoldásokat.

Elfogadási kritérium

Alapvetően két egymással összefüggő kérdést kell megválaszolnunk:

- ***Egy jobb megoldást mindig el kell fogadni?***
- ***Időnként elfogadható rosszabb megoldás is?***

Az elfogadási kritérium különbözteti meg a legnagyobb mértékben egymástól a különböző lokális keresési algoritmusokat.

Szimulált hűtés
Simulated Annealing (SA)

Szimulált hűtés (SA)

- Az SA nagyszámú iterációból áll.
- Egy közbenső k. iterációban s_k jelenti az aktuális megoldást.
- $s*$ jelenti az addig megtalált legjobb megoldást.
- Jelölje $f(s_k)$ az s_k megoldás célfüggvény értékét.
- A célfüggvény értékek alapján dönt, hogy két vizsgált megoldás közül melyiket tekinti jobb megoldásnak.
- A k. iterációban az s_k szomszédságából választ egy jelölt s_c megoldást.
- Ha s_c jobb megoldás mint s_k, akkor az lesz a következő iterációban a szomszédság kiindulási megoldása (bázisa) $s_{k+1}= s_c$. Ellenkező esetben a következő valószínűséggel fogadja el az s_c megoldást:

$$P(s_k, s_c) = e^{\frac{f(s_k)-f(sc)}{t_k}}$$

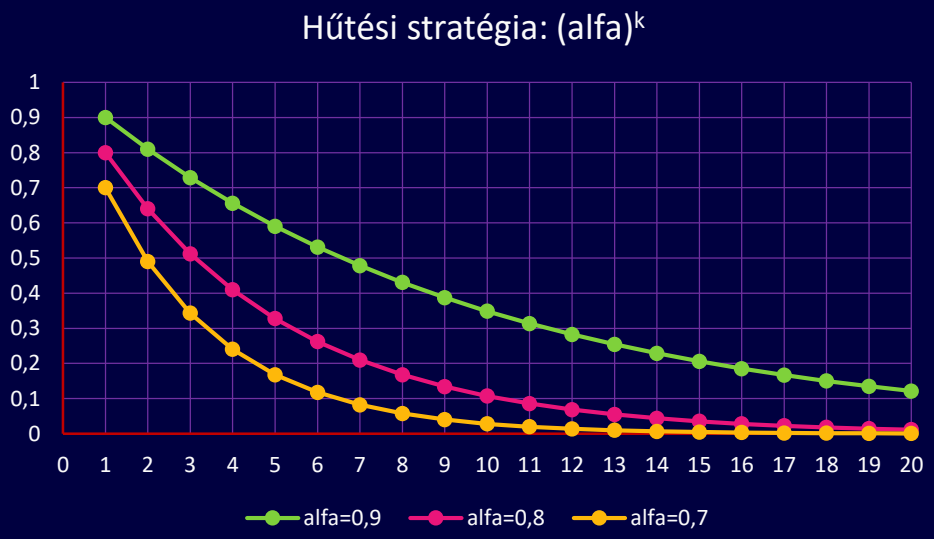
Hűtési paraméter

- A t_k egy vezérlő paraméter („hőmérséklet”):

$$t_1 \geq t_2 \geq \ldots \geq t_k \geq \ldots > 0$$

- A t_k értékét gyakran úgy választják meg, hogy

$t_k = a^k$, ahol az alfa 0 és 1 közötti valós szám.



Szimulált hűtés (SA) elfogadási kritériuma

$$P(s_k, s_c) = e^{\frac{f(s_k)-f(s_c)}{t_k}},$$

- Az SA a rosszabb megoldást is elfogadja esetenként. Ennek az a célja, hogy ***a lokális optimumból tovább tudjon lépni*** és később jobb megoldást is képes legyen megtalálni.
- Az elfogadási kritérium figyelembe veszi
  - ***a jelölt megoldás jóságát***, és ha az nagyon rossz, akkor nagyon kicsi valószínűséggel fogadja el.
  - **a hűtési paraméter értékét**, ezáltal a hűtési stratégiával befolyásolható a rosszabb jelölt elfogadása a keresés során (pl. a keresés elején nagyobb valószínűséggel fogadja el, mint a végén).

Szimulált hűtés (SA) leállási feltétel

- Sokféle ***leállási feltétel*** használható.
  - Az egyik lehetőség, hogy ha egy előre beállított értéket elér az iterációk száma, akkor az SA leáll.
  - Egy másik lehetőség, hogy a javulásokat figyeli és ha az utolsó javulástól számított előre megadott számú iteráción keresztül nincs javulást akkor leáll.

Szimulált hűtés (SA)

- A szimulált hűtés (Simulated Annealing, SA) algoritmus működése a következőképpen foglalható össze:
- 0. Lépés: Inicializálás.
- 1. lépés: Jelölt választása.
- 2. lépés: Vizsgálat.
- 3. lépés: Legjobb megoldás aktualizálása.
- 4. lépés: Következő iteráció/Kilépés.

SA: 0. lépés

Inicializáljuk a számlálót nullával: $k = 0$.

Készítsünk egy s_0 kiindulási megoldást.

Legyen $s_k = s_0$ és $s^* = s_0$;

Adjunk kezdeti értéket a t_k vezérlő paraméternek.

SA: 1. lépés

Válasszunk egy s_c szomszédos megoldást az s_k szomszédságából:

$$s_c \in N(s_k).$$

SA: 2. lépés

Vizsgáljuk meg az elfogadási kritériumot!

Ha $f(s_c) < f(s_k)$ akkor $s_{k+1} = s_c$.

Ha $f(s_c) \geq f(s_k)$

akkor generáljunk egy u_k véletlen valós számot egyenletes valószínűséggel a $[0, 1]$ intervallumból.

Ha $u_k \leq e^{\frac{f(s_k)-f(s_c)}{t_k}}$ akkor

$s_{k+1} = s_c$ egyébként $s_{k+1} = s_k$.

SA: 3. lépés

- Hasonlítsuk össze az aktuális megoldást az eddigi legjobb megoldással!

Ha $f(s_c) < f(s^*)$ akkor $s^* = s_c$.

SA: 4. lépés

Vegyük a következő iterációt!

$$k = k + 1.$$

Adjunk új értéket a vezérlő paraméternek úgy, hogy $t_{k+1} \leq t_k$ teljesüljön.

- Vizsgáljuk meg a leállási feltételt, ha az teljesül, akkor készen vagyunk, egyébként folytassuk az 1. lépéssel.

Megjegyzés

- Az SA algoritmus hatékonysága nagymértékben függ
  - a szomszédság megválasztásától és
  - a vezérlő paraméter értékének változtatási stratégiájától („hűtési stratégia").

Tabu keresés
Tabu Search (TS)

A tabu lista szerepe

- A TS sok szempontból hasonlít a szimulált hűtéshez. A TS is lokális keresési elvet követ.
- Az alapvető különbség az, hogy a TS ***determinisztikus elfogadási kritérium***ot használ.
- A TS a keresés minden iterációjában nyilvántart egy ***tabu listát***.
- A tabu lista olyan ***változásokat tartalmaz***, amelyek a korábbi lépésekben már megvizsgált megoldásokhoz vezetnek. Ezek tiltott változtatások.

Egy egyszerű példa

- ***Egy vektorral reprezentálható ütemterv*** esetén például a módosítás két munka felcserélését jelenti: (i,j) .
- Ha végrehajtásra került ez a módosítás egy adott iterációban, akkor ennek az inverze kerül fel a tabu listára: (j,i).
- Amíg a lista tartalmazza ezt a tiltott módosítást, addig a keresés során későbbi iterációkban nem engedélyezett ennek az inverz módosításnak az elvégzése mert akkor visszajuthatunk a korábbi megoldáshoz.

A tabu lista szerepe

- A tabu lista segítségével ***a lokális minimumból tovább tud lépni*** az algoritmus, mert a tiltott módosításokat nem engedi elfogadni, így nem következik be rövid ciklusú ismétlődés.
- A tabu lista folyamatosan frissül, a nyilvántartott módosítások száma ***korlátozott***.
  - Az új elem a lista elejére kerül és
  - a legrégebbi elem a lista végéről törlődik.

Tiltott megoldások nyilvántartása

- Léteznek olyan tabu keresési elven működő algoritmusok is, amelyek nem a tiltott módosításokat, hanem ***a tiltott megoldásokat*** tárolják a tabu listán.
- Ez akkor célszerű, ha a szomszédság definíciójában többféle módosító operátor együttesen is szerepelhet. Ilyenkor egy jelölt megoldás csak akkor fogadható el, ha az nem szerepel a tabu listán.

A tabu lista „mérete”

- A ***tabu lista elemszáma*** fontos paramétere a TS algoritmusnak.
  - Ha a maximális elemszám túl nagy, akkor a keresés nagyon korlátozottá válik,
  - ha pedig túl kicsi, akkor ismétlődő keresési útvonal alakulhat ki.

A tabu lista „szerkezete”

- Az ütemezési feladatok jellegétől függően ***a tiltások (tabu elemek) nyilvántartása*** nagyon különböző lehet.
- Tabu elem lehet:
  - Tiltott módosítás: pl. egygépes, vagy előzés nélküli Flow Shop feladat esetén az s vektorban (i,j) munkák cseréje → (j,i) tabu elem kerül a listára.
  - Tiltott megoldás: pl. kódolt teljes ütemterv (s) kerül a listára Flexible Job Shop, Open Shop, General Shop stb. feladat esetén. A tabu elem szerkezete függ a feladat típusától. Speciális kódolás → gyorsabb összehasonlítás.

Tabu keresés (TS)

- A TS algoritmus működése a következőképpen foglalható össze:
- 0. Lépés: Inicializálás.
- 1. lépés: Jelölt választása.
- 2. lépés: Vizsgálat.
- 3. lépés: Legjobb megoldás aktualizálása.
- 4. lépés: Következő iteráció/Kilépés.

TS: 0. lépés

Inicializáljuk a számlálót nullával: $k = 0$.

Legyen a tabu lista kezdetben üres!

Készítsünk egy s_0 kiindulási megoldást.

Legyen $s_k = s_0$ és $s^* = s_0$;

TS: 1. lépés

Válasszunk egy s_c szomszédos megoldást az s_k szomszédságából:

$$s_c \in N(s_k).$$

TS: 2. lépés

- Vizsgáljuk meg az elfogadási kritériumot!
- Ha az $s_k \rightarrow s_c$ módosítást nem tiltja egyetlen bejegyzés sem a tabu listán, akkor $s_{k+1} = s_c$ és az elvégzett módosítás inverze felkerül a tabu lista elejére, valamint ha a tabu lista elemszáma elérte a maximális értéket akkor az utolsó elemet töröljük.
- Ha az $s_k \rightarrow s_c$ módosítást tiltja valamelyik bejegyzés a tabu listán, akkor $s_{k+1} = s_k$ és folytassuk a 4. lépéssel.

TS: 3. lépés

- Hasonlítsuk össze az aktuális megoldást az eddigi legjobb megoldással!

Ha $f(s_c) < f(s^*)$ akkor $s^* = s_c$.

TS: 4. lépés

Vegyük a következő iterációt!

$$k = k + 1.$$

- Vizsgáljuk meg a leállási feltételt, ha az teljesül, akkor készen vagyunk, egyébként folytassuk az 1. lépéssel.

Megjegyzés

- A TS algoritmus hatékonysága nagymértékben függ
  - a szomszédság megválasztásától és
  - a tabu lista kezelési stratégiától.

Összefoglalás

1. Heurisztikus megoldási módszerek
2. A lokális szomszédsági keresés alapjai
3. Szimulált hűtés alkalmazása
4. Tabu keresés alkalmása

Köszönöm a figyelmet!

Az előadásvázlat elérhető az alábbi webcímen:

http://ait.iit.uni-miskolc.hu/~kulcsar/serv07.htm