1. Online és dinamikus problémák

1.1. Online problémák

A gyakorlati problémákban gyakran fordulnak elő olyan optimalizálási feladatok, ahol a bemenetet (más néven inputot, vagyis a feladatot definiáló számadatokat) csak részenként ismerjük meg, és a döntéseinket a már megkapott információk alapján, a további adatok ismerete nélkül kell meghoznunk.

Ilyen feladatok esetén *on-line problémáról* beszélünk. Az on-line algoritmusok elméletének igen sok alkalmazása van a számítástudomány, a közgazdaságtan, és az operációkutatás különböző területein.

Az on-line algoritmusok elméletének területéről az első eredmények az 1970-es évekből származnak, majd a 90-es évek elejétől kezdve egyre több kutató kezdett el az on-line algoritmusok területéhez kapcsolódó problémákkal foglalkozni. Számos részterület alakult ki és napjainkban is a legfontosabb, algoritmusokkal foglalkozó konferenciákon rendszeresen ismertetnek új eredményeket ezen témakörből. Ennek a jegyzetnek nem célja a témakör részletes áttekintése, terjedelmi okokból ez nem is lenne lehetséges ezen keretek között, további eredmények találhatóak a [6], [8], [11] művekben. Célunk néhány részterület részletesebb ismertetésén keresztül a legfontosabb algoritmustervezési technikák és bizonyítási módszerek bemutatása.

1.2. Elemzési módszerek

Mivel egy online algoritmusnak részenként kell meghozni a döntéseit a teljes bemenet ismerete nélkül, ezért egy ilyen algoritmustól nem várhatjuk el, hogy a teljes információval rendelkező algoritmusok által megkapható optimális megoldást szolgáltassa. Azon algoritmusokat, amelyek ismerik a teljes inputot off-line algoritmusoknak nevezzük.

Az online algoritmusok hatékonyságának vizsgálatára két alapvető módszert használnak. Az egyik lehetőség az átlagos eset elemzése, ebben az esetben fel kell tételeznünk valamilyen valószínűségi eloszlást a lehetséges bemenetek terén, és a célfüggvénynek az erre az eloszlásra vonatkozó várható értékét vizsgáljuk. Ezen megközelítés hátránya, hogy általában nincs információnk arról, hogy a lehetséges inputok milyen valószínűségi eloszlást követnek. Amennyiben elérhetőek való adatok, akkor az átlagos eset elemzés helyettesíthető az algoritmusok empirikus analízisével, azaz a valós tesztadatokon elért eredmények összehasonításával. Ilyen esetekben nincs szükségünk

az input eloszlások ismeretére, és a várható érték számítására sem.

A másik megközelítés egy legrosszabb-eset elemzés, amelyet versenyképességi elemzésnek nevezünk. Ebben az esetben az online algoritmus által kapott megoldás célfüggvényértékét hasonlítjuk össze az optimális off-line célfüggvényértékkel.

Egy online minimalizálási probléma esetén egy online algoritmust C-versenyképesnek nevezünk, ha tetszőleges inputra teljesül, hogy az algoritmus által kapott megoldás költsége nem nagyobb mint az optimális off-line költség C-szerese. Egy algoritmus versenyképességi hányadosa a legkisebb olyan C szám, amelyre az algoritmus C-versenyképes.

Általában egy tetszőleges ALG online algoritmusra az I inputon felvett célfüggvényértéket $\mathrm{ALG}(I)$-vel jelöljük. Az I inputon felvett optimális off-line célfüggvényértéket $\mathrm{OPT}(I)$-vel jelöljük. Használva ezt a jelölésrendszert a versenyképességet minimalizálási problémákra a következőképpen definiálhatjuk.

Az ALG algoritmus C-versenyképes ha $\mathrm{ALG}(I) \leq C \cdot \mathrm{OPT}(I)$ teljesül minden I input esetén.

Szokás használni a versenyképesség további két változatát. Egy minimalizálási probléma esetén az ALG algoritmus *enyhén C-versenyképes* ha van olyan B konstans, hogy $\mathrm{ALG}(I) \leq C \cdot \mathrm{OPT}(I) + B$ teljesül minden I input esetén.

Egy algoritmus enyhe versenyképességi hányadosa a legkisebb olyan C szám, amelyre az algoritmus enyhén C-versenyképes.

Természetesen igaz, hogy ha egy algoritmus erősen versenyképes valamely C konstanssal, akkor ezzel egyidejűleg ugyanezzel a konstanssal gyengén is versenyképes.

A fentiekben a minimalizálási problémákra definiáltuk a versenyképességi analízis fogalmait. A definíciók hasonlóan értelmezhetőek maximalizálási problémák esetén is. Ekkor az ALG algoritmus C-versenyképes ha $\mathrm{ALG}(I) \geq C \cdot \mathrm{OPT}(I)$ teljesül minden I input esetén, illetve enyhén C-versenyképes ha valamely B konstans mellett $\mathrm{ALG}(I) \geq C \cdot \mathrm{OPT}(I) + B$ teljesül minden I inputra. Tehát amíg mimimalizálandó célfüggvény esetén az előbbi konstansra $C \geq 1$, addig maximalizálandó célfüggvény esetén pedig $C \leq 1$.

1.3. Online problémák a szállítmánytervezésben

A szállítási problémák on-line jellege több esetben is előfordulhat. Egyrészt sok esetben a szállítás folyamata során keletkeznek új igények, amelyeket szintén ki kell szolgálnunk. Bizonyos alkalmazásokban megvárható, amíg az eddigi tervek szerinti végrehajtásban keletkezik szabad erőforrás, más esetekben az új kérés nem várhat hosszan és azonnal újratervezés szükséges. Egy másik online jelleg abból adódik, hogy a szállítás folyamat közben változhatnak a körülmények: változhatnak az úthálózat paraméterei, továbbá a szállítási tervhez képest keletkező egyéb eltérések is szükségessé tehetik az előzetes tervek megváltoztatását. A szállítmánytervezés területén belül szokás az online feladatokat dinamikus problémának is nevezni (ld. [3] és [14]).

A megváltozott körülmények kezelésére két alapvető stratégiát használunk. Az egyik stratégia befejezi a közvetlenül nem érintett szállítási terveket és csak a közvetlenül érintett tervek útvonalait módosítja, a másik megközelítés a tervek összességét újraoptimalizálja a megváltozott körülményeknek megfelelően. Ezeknek a stratégiáknak és több mindkét stratégiát használó algoritmusnak a vizsgálata jelenleg is folyik.

Általában a valós alkalmazásokban az input adatokkal kapcsolatosan két kérdés merül fel, egyrészt azoknak a lényeges változása, evolúciója amely teljesen új kérések megjelenését tekinti, a másik az input adatok minősége, ami azok stabilitását jellemzi. A minőséget általában sztochasztikus modellekkel jellemzik, ezzel ebben a tanulmányban nem foglalkozunk. Azzal kapcsolatban, hogy mennyire dinamikus az input különböző mérőszámokat vezettek be. A dinamikusság által okozott nehézség tulajdonképpen két tulajdonságtól függ, az egyik az, hogy mekkora az új adatok mennyisége, a másik az, hogy mennyire sürgős az azokban megjelent kérések kiszolgálása. Az első mérőszám a dinamizmus foka, amit a $\delta = n_{din}/n_{tot}$ formulával definiálhatunk, ahol n_{din} a dinamikus, később megjelent kérések száma, n_{tot} pedig az összes kérés száma. A fő különbség az online algoritmusok klasszikus területéhez képest, hogy a dinamikus szállítmánytervezési esetben gyakran olyan problémákat vizsgálnak, ahol a dinamizmus foka viszonylag alacsony, míg ilyen kikötések az online algoritmusoknál nem szokásosak. Amennyiben a dinamikus kérések sürgősségét is figyelembe vesszük, akkor számít, hogy egy kérés mikor válik ismerté, ezt az i kérésre az r_i érkezési idő adja meg. Amennyiben T jelöli a teljes szállítás végrehajtási idejét, D pedig a dinamikusan érkezett kérések halmazát, akkor használhatjuk a következő sürgősséget is figyelembe

vevő mérőszámot, amit a dinamizmus effektív fokának neveznek

$$\delta^e = \frac{1}{n_{tot}} \sum_{i \in D} \frac{r_i}{T}.$$

Fontosnak tartjuk megegyezni, hogy amennyiben időablakok is vannak a kérések teljesítéséhez, akkor a sürgősség nem a teljes szállítás végrehajtási idejétől függ, hanem az aktuális igény engedélyezett időablakának a végétől. Legyen c_i az i-dik kérés időablakának a befejezési időpontja. Erre a modellre a dinamizmus effektív fokát a következőképpen terjesztették ki.

$$\delta_{TW}{}^e = \frac{1}{n_{tot}} \sum_{i \in D} (1 - \frac{c_i - r_i}{T}).$$

2. Online utazó ügynök modellek

Ebben a fejezetben a szállítási útvonaltervezés azt a változatát foglaljuk össze, amelyben az igények on-line érkeznek továbbá csak az útvonaltervezés problémáját vesszük, így a szállítóeszközök kapacitását nem vesszük figyelembe. Az algoritmusok közvetlenül használhatóak abban az esetben ha a szállítandó mennyiségek kicsik, nem haladhatják meg a szállítóeszközök kapacitását, továbbá az itt tervezett algoritmusok továbbfejlesztései használhatóak lesznek majd a szállítmánytervezés on-line változatának megoldása során. A következő modelleket mutatjuk be.

2.1. On-line utazó ügynök probléma

Adott valamennyi pont a gráfban. A szállítóeszköz az útját egy meghatározott O pontban kezdi. Különböző időpontokban egy-egy címet kap - a célállomások címét -, ahová még el kell menjen. Ebben a pillanatban újratervezheti az útját úgy, hogy az összes címre, aminek a címét megkapta, eljusson. Ez a probléma a szállítmányszervezés speciális eseteiben fordulhat elő, amikor a szállítóeszköz begyűjtési vagy szétosztási feladatot hajt végre. A probléma on-line utazó ügynök feladat (OLTSP) néven ismert több változatát vizsgálták attól függően, hogy milyen megszorításokat alkalmazunk a feladat kiírásában (ld. [1], [5],[7]) . A legfontosabb a következő két változat. Az elsőben az eszköztől nincs megkövetelve az, hogy a kiindulási pontba visszatérjen miután minden hozzá beérkező kérést kielégített, a másik

megközelítésben viszont ez elvárt követelmény. Az első megközelítést nomád változatnak hívjuk és N-OLTSP-vel (Nomadic-OLTSP) jelölik, a másodikat pedig hazajáró TSP-nek nevezzük és H-OLTSP-vel (Homing-OLTSP) jelölik. Mindkét megközelítésben a cél a teljesítési idő minimalizálása.

Az N-OLTSP probléma megoldására kifejlesztették a következő algoritmust, amely egy mohó jellegű stratégiát követ. Minden alkalommal, amikor új kérés érkezik, tervezzük újra az utat úgy, hogy az a még ki nem szolgált kérések halmazára fektetett legrövidebb Hamilton út legyen.

Legyen $S(t)$ a t időpontig beérkezett még nem kielégített igények halmaza, beleértve a legújabb kérést is és a kiindulási pontot. Tegyük fel, hogy a t időpontban, amikor az új kérés beérkezett, a jármű a $p(t)$ pontban van. Tervezzük meg azt az útvonalat, amely $p(t)$-ből kiindulva minimális idő alatt kielégíti az $S(t)$-ben szereplő igényeket és folytassa a szállítóeszköz ezen útvonalterv alapján az elosztást.

Ezt az algoritmust GTR algoritmusnak nevezik és a versenyképességére az alábbi állítás teljesül:

1. Tétel. *A GTR algoritmus 2-versenyképes a nomád online TSP feladatra.*

Fontos megjegyeznünk, hogy a fentiekben leírt algoritmusok versenyképessége közel optimális, mivel teljesül az alábbi állítás.

2. Tétel. *A nomád online TSP feladatra nincs olyan online algoritmus, amelynek kisebb a versenyképessége, mint 2.*

Az N-OLTSP-re bemutatott mohó algoritmus könnyen átalakítható egy, a H-OLTSP-t megoldó mohó algoritmussá, annyit kell tenni, hogy nem utakat, hanem a kiindulási pontban végződő utakat kell keresni. Másrészt a H-OLTSP esetében egy mohó keretrendszeren belül felhasználható az a tény, hogy legvégül vissza kell érnünk a kiindulási pontba. Ezt úgy érjük el, hogy különbséget teszünk azon kérések között, amelyek viszonylag közel vannak a kiindulási ponthoz, és azok között, amelyek viszonylag messze vannak tőle. Ezt a következő PAH (Plan at Home) algoritmussal oldhatjuk meg.

1. Amikor a szerver a kiindulási pontban van, akkor azt az optimális utat kezdi el követni, amely optimálisan kiszolgálja az összes még ki nem szolgált kérést, majd visszatér az eredeti, kiindulási pontba.

2. Ha a t időpontban új kérés érkezik be a szállítóeszközhöz az x pontból, akkor az alábbi két lépés egyikét fogja tenni:

(a) Ha $d(x,o) > d(p(t),o)$, akkor a szerver visszamegy a kiindulási pontba (a legrövidebb úton), és az 1-es esetben leírtakat teszi.

(b) Ha $d(x,o) \leq d(p,o)$, akkor a szállítóeszköz a kérést addig figyelmen kívül hagyja, amíg újra vissza nem érkezik a kiindulási pontba.

3. Tétel. *A PAH algoritmus 2-versenyképes a hazajáró modellben.*

Ez az algoritmus optimális abban az értelemben, hogy nincs kisebb versenyképeséggel rendelkező online algoritmus, amint azt az alábbi állítás mutatja.

4. Tétel. *A hazajáró online TSP feladatra nincs olyan online algoritmus, amelynek kisebb a versenyképessége, mint 2.*

Mindkét algoritmusban az újraoptimalizálás során egy NP-nehéz feladatot kell megoldanunk, ezért a fenti algoritmusok futási ideje exponenciális. Az algoritmusok egy gyorsított polinomiális idejű változatát kapjuk, ha az újraoptimalizálási részben az egzakt megoldó algoritmusokat az utazó ügynök problémára széles körben vizsgált során heurisztikus eljárásokkal helyettesítjük.

2.2. Fuvarrendelési modellek

A szállítmánytervezés során az igények többnyire nem abban a formában érkeznek, hogy a szállítóeszköznek el kell mennie egy adott pontba, hanem abban a formában érkeznek, hogy a szállítóeszköznek el kell mennie egy pontba és onnan egy másik pontba kell szállítania. Tehát ebben a modellben a szállítóeszköz az útját egy meghatározott O pontban kezdi és oda is kell visszatérjen. Különböző időpontokban egy-egy címpárt kap, egy szállítás kezdő és végállomását. Ebben a pillanatban újratervezheti az útját úgy, hogy az összes szállítást, aminek az igényét megkapta végrehajthassa. Ez a probléma a szállítmányszervezés azon esetében fordul elő, amikor a szállítóeszköznek az egyes megrendeléseket külön-külön kell végrehajtania, nem szállíthat párhuzamosan különböző igényekhez tartozó értékeket egyszerre. Az így kapott on-line problémát on-line fuvarrendelési feladatnak nevezik a szakirodalomban (ld. [2],[9],[10]). A feladat megoldására a következő algoritmusokat fejlesztették ki:

Újratervező algoritmus: Amennyiben egy új kérés érkezik, a szállítóeszköz befejezi az aktuális szállítást, majd az eddig tervezett útvonaltervet elhagyja és egy új optimális útvonaltervet dolgoz ki, az aktuális még ki nem elégített igények figyelembevételével.

Ignoráló algoritmus: Ez az algoritmus ignorálja az új kéréseket mindaddig, amíg be nem fejezi az aktuális körutat amivel visszatér a kiindulási pontba. Utána az aktuális (már megjelent, de még nem teljesített) igények alapján az algoritmus kiszámít egy optimális körutat és ez lesz az új aktuális körút.

Ezen algoritmusok versenyképességét határozza meg az alábbi tétel.

5. Tétel. *Mind az újratervező mind az ignoráló algoritmus 5/2 versenyképes a fuvarrendelési modellben függetlenül attól, hogy a szállítóeszköznek mekkora a kapacitása.*

Amennyiben a szállítóeszköz kapacitása végtelen, azaz tetszőleges számú kérést szolgálhat ki egyszerre, akkor a PAH algoritmus következő kiterjesztése egy hatékony megoldó algoritmus. Ezt az algoritmust TIR (Temporaly Ignore Request) algoritmusnak nevezték el.

Megfontolt (SMART) algoritmus: Az algoritmus futása során a szállítóeszköz a következő három állapotban lehet.

- üres: Ebben az állapotban a szállítóeszköz a kiindulási O pontban van, és nincs olyan igény, amelyet ki kellene szolgálnia.
- alvó: Ebben az állapotban vannak igények, amiket ki kellene szolgálni, de ezek azonnali kiszolgálása túl költséges (a költségbecslő subrutin alapján), így az algoritmus még vár további igényekre.
- dolgozó: Ebben az állapotban az algoritmus egy (a költségbecslő subrutin által meghatározott) terv szerint szolgálja ki az igényeket.

A költségbecslő subrutin egy $\Theta > 1$ paramétertől függ és a következőképpen működik: Az algoritmus kiszámolja a minimális idejű S körutat, amely a kiindulási pontban kezdődik, kiszolgálja az összes aktuális igényt és a kiindulási pontban végződik. Amennyiben a körút befejezhető a Θt időpont előtt (t az aktuális időpont), akkor a subrutin az (S, dolgozó) értéket adja vissza, ellenkező esetben az (S, alvó) választ.

Az állapotokat az algoritmus a következőképpen váltogatja:

- Üres állapot: Amennyiben az algoritmus az üres állapotban van, akkor megvárja az első T időpontot, amelyben új igény merül fel. Ebben az időpontban meghívja a költségbecslő subrutint. Amennyiben az eredmény (S, dolgozó), akkor az eljárás a dolgozó állapotba kerül, a jármű elkezdi végrehajtani az S körutat. Ellenkező esetben az algoritmus az alvó állapotba kerül, egy Q ébredési idővel, amely a legkisebb T-nél nagyobb olyan érték, ami kielégíti a $T + l(S) \leq \Theta T$ feltételt (ahol $l(S)$ a körút végrehajtásának ideje).

- Alvó állapot: Az algoritmus az alvó állapotban az ébredési idejéig semmit nem csinál. Az ébredési időben újra meghívja a költségbecslő subrutint. Amennyiben az eredmény (S, dolgozó), akkor az eljárás a dolgozó állapotba kerül, a jármű elkezdi végrehajtani az S körutat. Ellenkező esetben az algoritmus az alvó állapotba kerül, egy újraszámított ébredési idővel.

- Dolgozó állapot: A dolgozó állapot alatt, amíg az aktuális körutat be nem járja, a jármű a hívásokat figyelmen kívül hagyja. Miután a jármű visszaért a kiindulási pontba vagy átáll az üres állapotba (ha nincs kielégítetlen igény) vagy meghívja a költségbecslő subrutint, és az eredménytől függően újra dolgozó állapotba vagy alvó állapotba kerül.

6. Tétel. *A SMART algoritmus 2-versenyképes.*

Mindhárom prezentált algoritmus során meg kell oldani a probléma offline változatát, mivel ez egy NP nehéz feladat a gyakorlatban használhatunk heurisztikák vagy közelítő algoritmusok által kapott megoldásokat is.

2.3. Online utazó szerelő probléma

Amennyiben a célfüggvény a városokba való érkezési idők összege illetve átlaga, akkor az on-line utazó szerelő problémáról beszélünk. Ezt a problémát a [13, 9] cikkekben vizsgálták. A feladat igen nehéz a szakirodalomban főleg csak egyenes esetén vizsgálták az online problémát. Két esetet különböztethetünk meg a klasszikus utazó szerelő probléma esetén a kérések egy pontként jelennek meg és a szervernek oda kell mennie a kiszolgáláshoz, és a megérkezésének az időpontját tekintjük a kérés kiszolgálási idejének. Másrészt itt is szokás a fuvarrendelési modellek vizsgálata is, ahol a kérés a metrikus tér

két pontja, és a kérés kiszolgálásához elsőként el kell mennünk a kérés kezdőpontjába, majd a végpontjába és a végpontba való megérkezéskor szolgáltuk ki a kérést. Itt figyelembe kell vennünk azt is, hogy van-e kapacitáskorlátja a szervernek. Amennyiben a szerver kapacitása 1, akkor egy kérés kiszolgálásának megkezdését követően azt a kezdőpontból el kell vinnie a végpontba, azaz egyszerre nem szolgálhatunk ki több kérést. Ha a szerver kapacítása végtelen, akkor egyszerre párhuzamosan több kérés kiszolgálása is folyhat. Ez azt jelenti, hogy egy kérés kiszolgálása az első olyan, a kérés megjelenését követő időpontban kezdődik, amikor a kezdőpontjába megy a szerver és az ezt követő első olyan időpontban fejeződik be, amikor a végpontjába megy a szerver. Ezen végtelen kapacitásos feladat megoldására az alábbi BCR (Blindly Construct Route) algoritmust javasolták a [9] cikkben.

BCR Algoritmus: Legyen $\sigma = \min\{t; |u|\}$, ahol u a 0 időpontban megjelent kérések kezdőpontjai közül a 0-hoz legközelebbinek az 0-tól való távolsága (amennyiben van ilyen kérés) és t az első nem 0 időpontban megjelenő kérés megjelenési ideje. A 0 időpontban a szerver elkezd jobbra mozogni, amíg a σ pontba meg nem érkezik. Ezt követően visszamegy 0-ba. Majd elmegy a másik irányba a -2σ pontig és onnan visszatér a 0-ba. Így folytatja a k-dik fázisban a $(-2)^k\sigma$ pontig megy majd onnan vissza 0-ba. Minden kérést akkor kezd kiszolgálni, amikor a kérés megjelenése után először átmegy a kezdőpontján, és akkor fejezi be a kiszolgálást, mikor ezt követően elsőként ér oda a kérés végpontjához.

7. Tétel. *([9]) Ha a metrikus tér egy egyenes, akkor a BCR algoritmus 9-versenyképes, ha minden kérésre ugyanaz a kezdő és végpont és 15-versenyképes az általános fuvarra hívó modellben.*

3. Többcélú változatok

Többcélú optimalizálásról beszélünk, amikor egy feladat megoldása során több, különböző szempontot is figyelembe kell venni, amelyek mindegyikét egy-egy a feladat lehetséges megoldásain értelmezett célfüggvény írja le. Bizonyos esetekben ilyenkor van egy kiemelt célfüggvény, amelynek a legnagyobb a jelentősége és a többi célfüggvény csak azon megoldások rangsorolására szolgál, amelyek a kiemelt célfüggvény szerint optimálisak. De az esetek többségében nincs ilyen rangsorolás a célfüggvények között, hanem mindegyik célfüggvényt egyformán, vagy hasonló mértékben figyelembe kell

vennünk. A probléma az, hogy ezekben az esetekben gyakran előfordul, hogy bizonyos megoldások, amelyek nagyon jó eredményt adnak az egyik célfüggvény szempontjából rosszul teljesítenek a másik célfüggvény alapján.

3.1. Többcélú optimalizálásban használt modellek

Tömören összefoglaljuk milyen megközelítéseket szokás használni a több célfüggvényt is figyelembe vevő modellekben, majd részletesebben bemutatjuk miként használhatóak ezek a hálózati folyamatok szintézisének esetén. Mivel a hálózati folyamatok témakörében többnyire minimalizálási feladatokat kell megoldanunk, ezért a továbbiakban feltételezzük, hogy minden szempont egy minimalizálandó célfüggvény által van megadva. Ennek megfelelően minimum feladatokra adjuk meg az összes definíciót. Mindezt az általánosság megszorítása nélkül feltehetjük, hiszen egy maximum feladat áttranszformálható egy minimum feladatra ha a célfüggvényt megszorozzuk -1-el. De megjegyezzük, hogy a definíciók maguk is minden nehézség nélkül kiterjeszthetőek maximalizálási feladatokra is.

Az alapvető megközelítés, hogy olyan megoldásokat keresünk, amelyek nem javíthatóak egyik célfüggvény szerint sem anélkül, hogy más célfüggvényekben rontanánk a megoldás hatékonyságán. A matematikailag precíz definíció érdekében tegyük fel, hogy k darab minimalizálandó célfüggvényünk van, amiket $f_1, \ldots, f_k$ jelöl, továbbá a probléma lehetséges megoldásainak a halmazát jelölje S.

Két megoldást összehasonlítva egy nyilvánvaló gondolat azt mondanunk, hogy egy $x \in S$ megoldás akkor jobb, mint egy $y \in S$, ha minden szempont szerint jobb, azaz minden $i = 1, \ldots, k$ értékre teljesül, hogy $f_i(x) < f_i(y)$. Ezen fogalom alapján definiálhatjuk a gyenge efficiens megoldásokat. Egy megoldás *gyenge efficiensnek* nevezünk, ha nincsen nála jobb megoldás. Tehát egy $x^* \in S$ megoldást akkor nevezünk gyenge efficiens megoldásnak, ha nincs olyan $x \in S$ megoldás, amelyre teljesülne $f_i(x) < f_i(x^*)$ minden $i = 1, \ldots, k$ esetén.

Egy másik, az előzőnél megengedőbb, és szélesebb körben használt fogalom két megoldás összehasonlítására a következő. Azt is mondhatjuk, hogy egy $x \in S$ megoldás akkor jobb, mint egy $y \in S$, ha egyik szempont szerint sem rosszabb és legalább az egyik szempont szerint jobb. Tehát minden $i = 1, \ldots, k$ értékre teljesül, hogy $f_i(x) \leq f_i(y)$ és van olyan $j \in \{1, \ldots, k\}$ index, amelyre $f_j(x) < f_j(y)$. Az ebben az értelemben vett legjobb megoldásokat szokás erősen efficiens vagy *Pareto optimális* megoldásoknak nevezni.

Tehát egy $x^* \in S$ megoldást akkor nevezünk Pareto optimális megoldásnak, ha nincs olyan $x \in S$ megoldás, amelyre teljesülne $f_i(x) \leq f_i(x^*)$ minden $i = 1, \ldots, k$ esetén és $f_j(x) < f_j(x^*)$ valamely $j \in \{1, \ldots, k\}$ indexre.

Többcélú optimalizálási feladatok esetén az egyik megközelítés az, hogy meghatározzuk a Pareto optimális megoldásokat. Másrészt az összes Pareto optimális megoldás legenerálásán kívül más módszereket is szokás használni többcélú optimalizálási feladatok megoldására. Az egyik lehetőség, hogy a különböző célfüggvényekből egy aggregált célfüggvényt hozzunk létre. Ennek a legegyszerűbb és legelterjedtebb módja az, hogy vesszük a célfüggvények egy pozitív súlyokkal képzett súlyozott összegét, és erre az aggregált függvényre keressük a minimális megoldást. Másrészt indokolt esetben más, minden változóban szigorúan monoton függvény is használható aggregált célfüggvényként. Így a problémát visszavezetjük egy egyetlen célfüggvényes optimalizálási feladatra. A két megközelítés közötti kapcsolatot adja meg az alábbi állítás.

8. Tétel. *Minden optimális megoldás amit egy, az eredeti célfüggvények pozitív súlyokkal vett lineáris kombinációjával képzett aggregált költségfüggvény alapján kaptunk Pareto optimális megoldása a többcélú optimalizálási feladatnak.*

Fontosnak tartjuk megjegyezni, hogy a fenti állítás teljesen hasonlóan igazolható minden olyan aggregált célfüggvényre, ami szigorúan monoton növekvő az összes változóban. Bizonyos alkalmazások esetén szoktak a lineáris kombinációnál bonyolultabb aggregált függvényeket is vizsgálni.

Egy további megközelítése a többcélú optimalizálási problémáknak, amelyben kiemelünk egy célt és ezen cél szerint keressük az optimális megoldást, miközben a többi szempontot korlátozási feltételként írjuk elő. Az egyszerűbb leírás érdekében tegyük fel, hogy az f_1 függvényt emeltük ki, amely szerint optimalizálni szeretnénk, ez a célfüggvények átjelölésével biztosítható. Ezt szokás ε-korlátozás módszerének is nevezni. Tehát egy korlátozásos egycélú redukciónál adottak $C_2, \ldots, C_k$ konstansok és az optimalizálási feladatunk a következő:

$$\begin{aligned} &\min f_1(x) \\ &f_i(x) \leq C_i, \qquad\qquad i = 2, \ldots, k \\ &x \in S \end{aligned}$$

Egy ilyen korlátozásos egycélú redukciónál nem feltétlenül teljesül, hogy az optimális megoldás az eredeti többcélú feladatnak Pareto optimális megoldása lesz, hiszen előfordulhatnak olyan megoldásai is a feladatnak, amelyek

f_1 szerint ugyanazt az értéket veszik fel, mint a kiválasztott optimális megoldás, de a többi célfüggvény szerint jobbak. Másrészt egy kicsit gyengébb állítást kimondhatunk.

9. Tétel. *Ha egy korlátozásos egycélú redukált feladatnak van optimális megoldása, akkor az az eredeti feladatnak gyenge efficiens megoldása lesz.*

3.2. Lehetséges célfüggvények szállítmánytervezési feladatoknál

Amennyiben egy több körutas szállítmánytervezési feladat van, akkor a körutak összhossza mellett egyéb függvényeket is szokás vizsgálni. Többcélú szállítmánytervezésről részletek találhatóak a [12] cikkben.

- Előfordulhat, hogy az éleknek két különböző költsége van. A legtöbb útvonal kereső szoftver esetén lehet minimalizálni megtett távolságra, időre és költségre is. Ezek hasonló mérőszámok de nem feltétlenül ugyanazok az optimális utak.

- A legkézenfekvőbb második költségfüggvény a körutak száma, szállítások esetén minden körút újabb szállítóeszközt és emberi erőforrást igényelhet, így nyilvánvalóan extra költséget jelent. Ezt gyakran beszámítják a célfüggvénybe, azaz aggregált függvényeket néznek. A további célfüggvények akkor érdekesek, ha adott a tervezendő körutak száma.

- Egy lehetséges célfüggvény a maximális körúthossz minimalizálása. Nyilván ez csak abban az esetben érdekes, ha adott a körutak száma (különben a megoldás) a depon kívül egyetlen pontot tartalmazó körutakat használna.

- Egy további célfüggvény a körutak egyensúlyozottsága. Ehhez definiálnunk kell a körutak terheltségét, ami lehet a hosszuk, de a kiszolgált kérések száma is figyelembe vehető. Ezt követően vesszük a legkisebb terheltségű körúttól való terhelési különbségét a többi körútnak és ezek összegét minimalizáljuk.

- Az egyes élekhez kockázati értékek is rendelhetők, amik a balesetek, késések valószínűségét adják meg. Ilyen esetekben cél lehet a kis kockázatú utak keresése is.

- Szokás olyan függvényeket is vizsgálni, amelyek nem csak az élektől függnek hanem a kérésektől. Ilyen modelleket akkor tekintenek, ha nem kell az összes kérést kiszolgálni. Ezeket a többcélú modelleket az alábbiakban tekintjük át.

3.3. Visszautasításos modellek

A visszautasításos vagy büntető TSP esetén minden ponthoz hozzátartozik még egy további érték. Ez egy π_i büntetés, amit azon pontok után kell fizetni, amelyeket nem látogattunk meg. Tehát a költségünk két részből adódik össze, egyrészt ki kell fizetnünk a büntetéseket minden pontra, amit nem látogattunk meg, másrészt a meglátogatott pontokat bejáró körútra venni kell a körút költségét, ami a szokott módon a körútban szereplő élek súlyainak összege. A feladat NP-nehéz, hiszen ha minden büntetés végtelen, akkor megkapjuk a TSP feladatot. Az alábbiakban bemutatunk egy érdekes közelítő megoldást adó algoritmust a [4] dolgozat alapján. Az algoritmus szimmetrikus és a háromszög egyenlőtenséget kielégítő költségmátrixok esetén működik. Az alapötlet az, hogy minden j-re definiáljuk a BTSP(j) feladatot, ami a fenti feladat azon megszorítás mellett, hogy a j pontot mindenképpen meg kell látogatnunk. Ez azért jó, mert a feladat optimális megoldása vagy ezen BTSP(j) feladatok megoldásai közül lesz a legjobb vagy az lesz, hogy minden kérést visszautasítunk. Tehát elegendő ezen BTSP(j) feladatok optimális megoldásait jól közelíteni.

Egy ilyen feladat felírható egészértékű programozási feladatként, ahol az x_e változók ($e \in E$) a kiválasztott éleket adják meg a szokott módon, a változó értéke 1 ha az e él benne van a körútban, és az y_j változók ($j \in N$) pedig a kiválasztott csúcsokat $y_i = 1$, ha az i várost meglátogattuk, 0 egyébként. Az egyszerű leírás érdekében használjuk a $\delta(S)$ jelölést, ami azokat az éleket jelöli, amelyek az S halmaz és a komplementere között mennek, például $\delta(\{i\})$ az i pontból kimenő élek. Ekkor a feladat

$$\begin{aligned}
&\textstyle\sum_{e\in E} c_e x_e + \sum_{i\in N}(1-y_i)\pi_i \rightarrow && \min \\
&\textstyle\sum_{e\in\delta(\{i\})} x_e = 2y_i && \forall i \in N \\
&\textstyle\sum_{e\in\delta(S)} x_e \geq 2y_i && \forall i \in N, S \subset N \text{ with } |S \cap \{i,j\}| = 1 \\
&y_j = 1
\end{aligned}$$

A célfüggvény helyessége adódik az x_e és y_i változók definíciója alapján. Az első feltétel garantálja, hogy ha egy pont y_i szerint benne van a körútban, akkor átmegy rajta körút, a második feltétel pedig a részkörút kizáró feltétel. Ez biztosítja, hogy ha egy i pont benne van a körútban, akkor minden olyan

S halmazból, ami i és j közül pontosan az egyiket tartalmazza kell kimenjen él. Az egészértékű felírás segítségével definiálhatjuk az alábbi közelítő algoritmust.

BGSW algoritmus:

- Minden j-re hajtsuk végre a következő lépéseket
  - Vegyük a BTSP(j) feladat egészértékű programozási felírásánal LP relaxációját, azaz azt a változatot, ahol x_e-ről és y_i-ről csak annyit kötünk ki, hogy a $[0, 1]$ intervallumban vannak és nem követeljük meg, hogy egészek.
  - Oldjuk meg ezt az LP relaxációt egy polinom idejű algoritmussal (pl belső pontos módszer).
  - Válasszuk ki azokat a pontokat, amelyekre az LP relaxáció megoldásában $y_i \geq 3/5$.
  - Az ezen pontokból álló TSP feladatra adjunk egy közelítő körutat a Christofedes heurisztikával, a többi kérést utasítsuk vissza. Legyen ez a megoldás M_j.
- Vegyük az M_j megoldások közül a legkisebb költségűt.
- Ha ennek a megoldásnak a költsége kisebb, mint minden kérés visszautasítása, akkor ezt adja vissza az algoritmus, egyébként minden kérést visszautasít.

Az algoritmus legrosszabb esetben is jól közelíti az optimumot, amint azt az alábbi állítás mutatja.

10. Tétel. *A BGSW algoritmus* 5/2*-approximációs.*

3.4. Díjgyűjtő modellek

A díjgyűjtő utazó ügynök modellben minden ponthoz hozzátartozik a büntetésen kívül még egy további érték. Ez egy w_i díj, amit akkor kapunk meg, ha a pontot az ügynök meglátogatta. Továbbá adott egy Q kvóta érték, ami a minimálisan összegyűjtendő díjak összegét adja meg. Tehát a lehetséges megoldások olyan körutak, amelyekben legalább Q értéknyi díjat összegyűjtünk, és ezek közül keressük a minimális költségűt. Egy adott körút költsége

két részből adódik össze. Vennünk kell az ügynök által megtett távolságnak (ami a körútban szereplő élek költségeinek összege) és a nem meglátogatott pontok büntetéseinek összegét. Ha minden büntetés 0, azaz a feladat csak egy minimális hosszúságú köút megtalálása, amely kielégíti a kvótára vonatkozó feltételt, akkor kvóta TSP-ről beszélünk.

Az online feladatban a kéréseknek van egy érkezési ideje is, amit a j-edik kérés esetén r_j jelöl és ezen időpont előtt nem lehet kiszolgálni a kérést. Itt is egy olyan körutat kell tennünk, amelyben a meglátogatott pontok díjaiból összegyűjtünk legalább Q értéket, és a cél a körút befejezési idejének és a nem meglátogatott pontok büntetéseinek összegének összege.

Az online feladat megoldására a WGR (wait and go with restart) algoritmust fejlesztették ki. Az alapötlete az, hogy igyekszik kivárni, amíg elegendő információt kap és ezáltal elkerülni a kezdeti rossz döntéseket. Két állapota van a várakozási és dolgozó állapotok. Kezdetben az algoritmus várakozási állapotban van, majd az alábbi szabályok szerint működik.

- Ha várakozási állapotban van, akkor vár addig a t időpontig, amelyre a t időpontig megérkezett kéréseke kiszolgálásának optimális offline költsége pontosan t, és ekkor átlép a dolgozó állapotba.

- Ha az algoritmus a t_D időpontban lép a dolgozó állapotba, akkor egyenes visszamegy az origoba. Utána kiszámolja a hátralevő kérések optimális megoldását és elkezdi azt a körutat bejárni. Ha befejezte újra a várakozó állapotba kerül. Továbá, ha dolgozó állapotban van és ezalatt egy új kérés érkezik egy t időpontban, akkor kiszámolja a t időpontig érkezett kérések kiszolgálásának optimális offline költségét, és ha ez nagyobb, mint t, akkor megáll és átlép várakozó állapotba.

Az algoritmus versenyképességét adja meg az alábbi állítás.

11. Tétel. *A WGR algoritmus* 7/3*-versenyképes az online díjgyűjtő TSP feladatra.*

4. Egyéb nem versenyképesség alapú dinamikus megközelítések

A versenyképességi elemzésben tekintett algoritmusok többnyire részfeladatként megoldanak NP-nehéz problémákat. Ezen elemzés során a futási időkkel nem foglalkozunk, így ezt nem tekintjük problémának, mint említettük

minden algoritmus kiterjeszthető olyan módon, hogy az optimális megoldás helyett valamilyen gyorsabb, közelítő módszert használunk. Amennyiben a közelítő módszerre adott egy legrosszabb approximácós korlát (mint például a fentiekben bemutatott büntetéses modellben használt algoritmus, vagy az alap TSP-nél használt Christofides algoritmus esetén, akkor általában ennek felhasználásával kaphatunk egy versenyképességi korlátot az online megoldásra is.

A gyakorlati feladatoknál fontos a számítást segítő algoritmusok futási ideje, hiszen egy módosított tervet csak akkor tudunk elkezdeni, ha végrehajtottuk a tervet elkészítő számításokat. Ennek megfelelően az újraoptimalizáláson alapuló dinamikus megoldó algoritmusokat két osztályba szokás sorolni.

Periodikus újraoptimalizálós algoritmusok: Ez a csoportja az algoritmusoknak a kézenfekvő megoldást választja. Vagy adott időközönként vagy minden új dinamikus kérés megjelenése esetén, veszi az aktuális állapotot (a szerver vagy szerverek állapota, a még nem teljesített kérések) és ezen inputra meghatároz egy statikus megoldó algoritmussal egy optimális megoldást. Ezt követően a következő újraoptimalizálásig ezt a megoldást követi. Ezen megközelítés előnye, hogy a széles körben kutatott statikus megoldó algoritmusok halmazából választhatunk egy algoritmust. A megközelítés hátránya, hogy az újraoptimalizálás a kezdetektől indul nem használjuk ki a már esetlegesen meglevő információkat, megoldáskezdeményeket.

Folytonos újraoptimalizálás: Ez a csoportja az algoritmusoknak igyekszik kihasználni a meglevő információkat. Az aktuális megoldáshoz használt segédinformációkat nem töröljük, hanem karbantartjuk, hogy később a dinamikusan érkezett új kérések miatt kicsit megváltozott input megoldásában segítségünkre lehessenek. Ezt a megközelítést jól használhatjuk különböző metaheurisztikáknál, mint a lokális kereső vagy genetikus algoritmusok. Számon tartjuk juó megoldásoknak egy halmazát, és amennyiben változik az input ezeket a megoldásokat megfelelően változtatjuk és innen kezdjük az újabb feladat megoldásainak keresését. Ezzel csökkenthetjük az újraoptimalizáló algoritmus futási idejét, és ezáltal javíthatunk a dinamikus változásokra való reagálásunk idején.

Hivatkozások

[1] G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, M. Talamo, Algorithms for the on-line traveling salesman. *Algorithmica* **29**, 560–581, 2001.

[2] N. Ascheuer, S.O. Krumke, J. Rambau, Online dial-a-ride problems: Minimizing the completion time. *In: Proceedings of the 17th STACS.Volume 1770 of Lecture Notes in Computer Science*, Springer 639–650, 2000.

[3] G. Berbeglia, J.F. Cordeau, G. Laporte, Dynamic pickup and delivery problems, European Journal of Operational Research 202 (2010) 8–15

[4] D. Bienstock, M. Goemans, D. Simchi-Levi, D. Williamson, A note on the prize collecting traveling salesman problem, *Mathematical Programming*, **59**, 413–420, 1993.

[5] M. Blom, S:O, Krumke, W.E. de Paepe, L. Stougie, The online TSP against fair adversaries, *INFORMS J. Computing* **13** 138–148, 2001.

[6] A. Borodin, R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, 1998.

[7] P. Damaschke, Two short notes on the on-line travelling salesman: handling times and lookahead. Theoretical Computer Science, 289 (2002), 845–852.

[8] Gy. Dósa, Cs. Imreh, *Online Algoritmusok*, online tankönyv, Typotex, 2012.

[9] Feuerstein, E., Stougie, L.: On-line single server dial-a-ride problems. Theoretical Computer Science 268 (2001), 91–105.

[10] Grötschel M., Krumke, S.O., Rambau, J, Online optimization of complex transportation systems. Online optimization of large scale systems, (2001) 705–729, Springer, Berlin, 2001.

[11] Cs. Imreh, Competitive analysis, In *Algorithms of Informatics Volume 1*, ed. Antal Iványi, mondAt, Budapest 2007, 395–428.

[12] N. Jozefowiez, F. Semet, E.G. Talbi, Multi-objective vehicle routing problems, European Journal of Operational Research 189 (2008) 293–309.

[13] Krumke, S.O., de Paepe, W. E., Poensgen, D., Stougie, L.: News from the online traveling repairman. Theoretical Computer Science 295 (2003), 279–294.

[14] V. Pillac, M. Gendreau, C. Gueret, A.L. Medaglia, A Review of Dynamic Vehicle Routing Problems, *European Journal of Operational Research*