Online algoritmusok 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 megváltozott körülmények kezelésére két alapvető stratégiát használnak.

- 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.

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 itt 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.

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

Online utazó ügynök probléma

Adott valamennyi pont egy metrikus térben (vagy egy 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. A legfontosabb a következő két változat.

- Az elsőben az eszköztől nincs megkövetelve az, hogy a kiindulási pontba visszatérjen miután minden hozzá beérkező kérést kielégített ezt nomád változatnak hívjuk és N-OLTSP-vel (Nomadic-OLTSP) jelöljük.

- A másik megközelítésben viszont a visszatérés elvárt követelmény, ezt hazajáró TSP-nek nevezzük és H-OLTSP-vel (Homing-OLTSP) jelölik.

Mindkét megközelítésben a cél a teljesítési idő minimalizálása.

GTR Algoritmus

Az algoritmus egy mohó újratervező algoritmus. 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:

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

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

Bizonyítás Az első kérés a 0 pontba jön. Utána várunk az 1 időpontig, és vagy a +1 vagy a -1 pontba érkezik egy kérés. Az optimális költség egy az online költség legalább 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.

Tétel A PAH algoritmus 2-versenyképes a hazajáró modellben.

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

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.

Lehetséges célfüggvények

Amennyiben egy több körutas szállítmánytervezési feladat van, akkor a körutak összhossza mellett egyéb függvényeket is szokás vizsgálni.

- 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.

Újratervezést segítő számítások

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 jó 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.

Egy valós idejű pozicionáló rendszer online problémai

A valós idejű pozicionáló rendszerek (RTLS) célja gyorsan és pontosan meghatározni bejövő jelekből különböző mozgó objektumok helyzetét. Rengeteg területen alkalmaznak ilyen rendszereket logisztika, biztonságtechnika, sport (Chip-in-the-Ball technológia).

A követendő objektum rendszeresen kiad jeleket, amiket begyűjtő csomópontok gyűjtenek. Ezek a csomópontok a kapott jeleket továbbküldik a számítást végző szervernek. A megoldandó feladat egy része egy klaszterezési probléma. Nem tudjuk mikor érdemes továbbküldeni a jeleket a pozició számításhoz. Ha túl gyorsan küldjük, akkor a későn érkező jeleket már nem használjuk, ami ront a pontosságon, ha túl korán, akkor már késve kapjuk a poziciót ami megint ront a rendszer pontosságán.

Matematikai model

Az input $X : x_1, \ldots, x_n$ a gyűjtött adatok:

- $Burst(x_i)$ jelöli azt, hogy a jelet mikor küldték (melyik ciklusban jött)
- $Node(x_i)$ adja meg, hogy melyik gyűjtő csomópont fogta a jelet
- $Rcpt(x_i)$ adja meg a jel érkezési idejét. A különbséget az ugyanazon ciklusba eső első és utolsó jel érkezési ideje között a ciklus hosszának nevezzük.
- $B_j = \{x_i | Burst(x_i) = j\}$ a j-edik ciklus jelei, ebből $B'_j \subseteq B_j$ az a részhalmaz amit továbbküldünk a pozicionálásra. A továbbküldés idejét jelöli p_j.

A csomópontonak különböző tulajdonságaik lehetnek, így mindegyik kap egy $w_k > 0$ fontossági értéket, feltesszük, hogy $w_1 = \min_{k=1}^{m} w_k$. Minden j-re és k csomópontra legyen $e_{jk} = 1$ ha k adatát beküldtük és 0 egyébként. Legyen

$$r_j = \max_{i=1}^{n} \{Rcpt(x_i) | x_i \in B'_j\}.$$

Ekkor a maximalizálandó célfüggvény

$$f = \sum_{j=1}^{b} \sum_{k=1}^{m} e_{jk} w_k - c \sum_{j=1}^{b} (p_j - r_j).$$

Tétel Nincs online algoritmus, aminek kisebb a versenyképessége, mint $\frac{\sum_{k=1}^{m} w_k}{w_1} \geq m$.

Bizonyítás Az első elem az első ciklusban az első csomóponthoz a 0 időpontban érkezik. Ha az algoritmus úgy dönt, hogy $p_1 \geq \frac{w_1}{c}$, akkor nem jön több jel, az online nyeresség legfeljebb 0 az offline pedig a $p_1^{opt} = 0$ értékkel w_1. Tehát ez a döntés nem vezet versenyképes algoritmushoz.

Feltehetjük, hogy $p_1 < \frac{w_1}{c}$. Ekkor $m-1$ további elem érkezik az első ciklusban a $2, \ldots, m$ csomópontokhoz a $\frac{w_1}{c}$ időpontban. Ekkor az online nyeresség legfeljebb w_1 az optimális algoritmus nyeressége a $p_1^{opt} = \frac{w_1}{c}$ értékkel $\sum_{k=1}^{m} w_k$.

A Nem Vár (NV) algoritmus, amint jön egy adat továbbküldi.

Tétel Az NV algoritmus versenyképességi hányadosa $\frac{\sum_{k=1}^{m} w_k}{w_1}$.

Korlátozott ciklushosszú model

Az alkalmazásban a ciklusok hossza nem lehet nagyon nagy, így feltehetjük, hogy ismert a maximális ciklushossz d. Feltesszük, hogy $\frac{w_1}{c} > d$ egyébként az előző esethez jutunk.

Tétel Nincs olyan algoritmus, aminek kisebb a versenyképessége, mint $\min\{\frac{w_1}{w_1 - c \cdot d}, \frac{\sum_{k=1}^{n} w_k}{w_1}\}$.

A konstans várakozási idő (KVI) algorimus, d ideig vár és akkor küldi tovább az adatokat.

Tétel A KVI algoritmus versenyképességi hányadosa $\frac{w_1}{w_1 - c \cdot d}$.

Váltakozó ideig váró (VIV) algoritmus

Az algoritmus tanulja a várakozási időt. Számos paraméter használ nem részletezzük.

Az algoritmus elemzéséhez a Fraunhofer Intézet szimulációs rendszerét használták. Virtuális objektum mozgott egy megadott pályán. Az alábbi véletlen döntéseket használta.

- az adatcsomagok egy adott valószínűséggel vesztek el
- két ciklus között az idő és a ciklus adatcsomagai is normális eloszlással lettek generálva.

A tesztek azt mutatták, hogy a KVI algoritmus hatékonysága nagymértékben függ az átlagos ciklushossztól. Minden paraméterre adott jó eredményeket de nagyon rosszakat is. Tehát ez az algoritmus csak akkor használható, ha van előzetes információ az átlagos ciklushosszról. A jó inputokon jó hatékonyságot ért el az optimális érték 90 százalékát.

A VIV algoritmus jó eredményt adott ismeretlen ciklushosszakra is. A teszteken elért legrosszabb hányadosa 0.914 volt, az átlagos hányados 0.95.

Szimulációs elemzések az ütemezésnél

A felhasznált munkák:

- Valódi munkák
  - MPP's (massziv párhuzamos processzorok) a Feitelson's Parallel Workloads Archive datbázisból.
  - munkák a Pittsburg Supercomputing Center's Cray C90 gépről.
  - munkák Sun Ultra munkállomásáról a Max Planck intézetnek.
- Valószínűségi eloszlások
  - standard eloszlások (uniform, ex- ponential, hyperexponential, Erlang).
  - Korlátozott Pareto.