Kétfázisú robusztus optimalizálás

Marosvári Ágnes

2021. május 14.

A kétfázisú optimalizálás egy olyan feladat, ahol egymás után kétszer is optimalizálunk. Az első fázisbeli döntéshozásnál még nem áll minden információ a rendelkezésünkre, az csak a döntés meghozása után derül ki, majd ezután az első fázisbeli döntés és az új információ függvényében hozzuk meg a második fázisbeli döntést. Az ismeretlen információhalmazt tekinthetjük egy véges halmaz vagy egy poliéder pontjainak. Mivel az erre felírt modell bizonytalanságot tartalmaz, de ezt nem a valószínűségszámítás eszközeivel szeretnénk kezelni, a megoldást robusztussá kell tenni, azaz biztosítani kell, hogy a bizonytalan halmaz bármely pontját véve megoldás marad. Ezen belül viszont a lehető legjobb megoldást szeretnénk megtalálni. Tehát például egy robusztus modellnek, amelyben minimalizálási feladatot szeretnénk megoldani, a célfüggvényében egy max min kifejezés fog állni, ahol a bizonytalan halmaz elemein vett maximalizálás jelenti, hogy a legrosszabb esetre készülünk fel, de ezen belül már minimalizálunk, és a legjobb megoldást keressük.
Ebben a félévben azzal folytattam a kétfázisú robusztus optimalizálással kapcsolatos munkát, hogy az előző félévben megismert C&CG algortimust implementáltam egy konkrét modell esetén, alaposan kielemeztem a paraméterek különböző értékei esetén kapott futási eredményeket, és alkalmaztam egy valóéletbeli problémára, ahol a problémát, illetve a kapott eredményeket részletesen vizualizáltam. A használt modellt az [1]-es cikkből vettem, és a programot FICO Xpress + Mosel környezetben implementáltam.

1. A modell

A modell egy p-median problémának a kétfázisú robusztus változata. Ebben egy hozzárendelési feladatot szeretnénk megoldani az I-vel jelölt kliens- és a J-vel jelölt gyárhalmaz között ($J \subseteq I$, sőt először még tegyük fel, hogy $I = J$). A $|J|$ darab gyár közül pontosan p darabot szeretnénk megnyitni, ezeknek nincsen megnyitási költsége. Ezután a megnyitott gyárakhoz rendeljük hozzá az I kliens halmazt. Innentől viszont eltér a modell a jól ismert p-median problémától, ugyanis azt a jelenséget is szeretnénk modellezni vele, ahol előre nem látható okoból néhány gyár leáll, ekkor pedig a megmaradt gyárakkal kell újratervezni a hozzárendeléseket.
A gyárakhoz tartozó bináris döntési változó $y \in \{0,1\}^{|J|}$ lesz, ahol $y_j = 1$ pontosan akkor, ha megnyitjuk a j-edik gyárat. Minden i klienshez tartozik egy d_i igény, amelyből ha egy egységet a j-edik gyár szolgál ki, akkor ennek a költsége c_{ij}. Előírjuk, hogy a költségfüggvény olyan legyen, hogy $c_{ii} = 0 \ \ \forall i$. Az első fázisnak van egy másik döntési változója is, $x \in [0,1]^{|I|\times|J|}$, ahol x_{ij} azt írja le, hogy i-edik kliens igényének hányad részét szolgálja ki a j-edik gyár (amelynek nyitottnak kell lennie).
Ezek után térjünk rá a modellben szereplő bizonytalanságra, amelynek értéke azután derül ki, hogy y és x változókat rögzítettük. Ehhez bevezetünk egy $z \in \{0,1\}^{|J|}$ bináris változót, ahol $z_j = 1$ pontosan akkor, ha a j-edik gyár kiesik. A kieső gyárak számát korlátozzuk egy előre rögzített k egész számmal. A kieső gyárak miatt több kliens igénye kiszolgáló nélkül marad. Ezeket a klienseket a megmaradt gyárakhoz kell hozzárendelnünk, ezért bevezetjük a második fázis döntési változóit, $w \in [0,1]^{|I|\times|J|}$-t és $q \in [0,1]^{|I|}$-t. w_{ij} fogja kifejezni azt, hogy a második fázisban az i-edik kliens igényének hányad részét szolgálja ki a j-edik gyár, míg q_i azt, hogy az i-edik kliens igényének mekkora része marad kiszolgálatlanul. Egy i klienst csak egy megnyitott és épen maradt gyárhoz rendelhetjük hozzá, amelynek az igény $1 - q_i$ részét kell

kiszolgálnia. Az eddig leírtakat az alábbi korlátok fejezik ki.

$$(1-\varrho)\min\sum_{i\in I}\sum_{j\in J}c_{ij}d_jx_{ij}+\varrho\max_{z}\min_{(w,q)\in S(y,z)}\left(\sum_{i\in I}\sum_{j\in J}c_{ij}(1-\theta z_i)d_iw_{ij}+\sum_{i\in I}M(1-\theta z_i)d_iq_i\right) \tag{1}$$

$$x_{ij}\le y_j\quad\forall i,j \tag{2}$$

$$\sum_i x_{ij}=1\quad\forall i \tag{3}$$

$$\sum_j y_j=p \tag{4}$$

$$y_j\in\{0,1\}\quad\forall j,\quad x_{ij}\ge 0\quad\forall i,j \tag{5}$$

$$\sum_j z_j\le k \tag{6}$$

$$z_j\in\{0,1\}\quad\forall j \tag{7}$$

Ahol $S(y,z)$ azokból a (w,q) vektorokból áll, amelyek teljesítik a következő korlátokat:

$$w_{ij}\le 1-z_j\quad\forall i,j \tag{8}$$

$$w_{ij}\le y_j\quad\forall i,j \tag{9}$$

$$\sum_j w_{ij}+q_i=1\quad\forall i \tag{10}$$

$$w_{ij}\ge 0\quad\forall i,j,\quad q_i\ge 0\quad\forall i \tag{11}$$

Ebben (1)-(5) fejezi ki a célfüggvényt és a p-median probléma feltételeit, (6)-(7) a bizonytalan halmazt, míg (8)-(11) az újrakonfigurálást.

A célfüggvény két tagból tevődik össze: a normál körülmények közötti működés költségének és a legrosszabb esetbeli működés költségének összege. A $\varrho\in[0,1]$ paraméterrel állíthatjuk be, hogy a tervezés során melyik esetet milyen súllyal szeretnénk figyelembe venni. Minél nagyobb a ϱ, annál konzervatívabb a modell. Az első tagban az első fázisbeli hozzárendeléssel kapott szállítási költség minimalizálása szerepel. A második tagban a lehetséges z vektorokra vett maximum fejezi ki, hogy a legrosszabb esetet keressük, és ezen belül minimalizáljuk a költséget, amely két dologból tevődik össze. A kiszolgált igények költségéből és a nem kiszolgált igényekből, amelyet egységenként egy M konstanssal büntetünk. Az itt szereplő θ paraméter azt a jelenséget kívánja modellezni, hogy amennyiben az i-edik telephelyen leáll egy gyár, ott a kliens igénye is megváltozhat. Például egy természeti katasztrófa esetén egy luxustermék iránti kereslet lecsökkenhet, míg a gyógyszerek iránti kereslet megnövekedhet. Ehhez a θ paramétert egy $(-\infty,1]$-beli valós számnak rögzítjük, és így látható, hogy ha a $(0,1]$ intervallumba esik, akkor az az igénycsökkenést fejezi ki, míg negatív számként az igény növekedését.

Megjegyzés. *A célfüggvény felírásánál kihasználtuk, hogy a I és J halmazok megegyeznek, azaz minden kliens egyben potenciális telephely is. Ha azonban a $J\subset I$ esetet szeretnénk vizsgálni, akkor azon klienseknél, amelyek az $I-J$ halmazba esnek, az $(1-\theta z_i)$ kifejezés nem értelmes, így itt például csinálhatjuk azt, hogy kettébontjuk a célfüggvényben szereplő összegzést a következőképpen.*

$$(1-\varrho)\min\ldots+\varrho\max_{z}\min_{(w,q)\in S(y,z)}\left(\sum_{i\in J}\sum_{j\in J}c_{ij}(1-\theta z_i)d_iw_{ij}+\sum_{i\in I-J}\sum_{j\in J}c_{ij}d_iw_{ij}+\sum_{i\in J}M(1-\theta z_i)d_iq_i+\sum_{i\in I-j}Md_iq_i\right)$$

Nem okoz tehát problémát, ha a modellt a $J \subset I$ esetre szeretnénk alkamazni, de azért, hogy a továbbiakban is tömören leírható legyen a célfüggvény, maradjon meg a feltevés, hogy $I = J$.

2. A C&CG algoritmus

Az előző félévben részletesen foglalkoztam az algoritmus elméleti hátterével. Röviden összefoglalva az ilyen kétfázisú modellek esetén a megoldás keresése nehéz, ezért nem a teljes problémát szeretnénk rögtön megoldani, hanem újabb és újabb iterációkban megkeressük a bizonytalanságot adó poliéder extrém pontjait (szignifikáns szcenáriók). Ezt egy mesterprobléma–szubprobléma szerkezetű algoritmussal tesszük meg. Kezdetben tehát kiindulunk az első fázis feltételeiből (a modell (2)-(5) korlátjai), megoldjuk, és kapunk egy optimális (y, x) párt, amelyből egy alsó korlátot tudunk előállítani (hiszen az eredeti minimalizálási feladat egy relaxáltját oldottuk meg). Ezután következik a szubprobléma, ahol kiszámoljuk, hogy ezen y vektor mellett a második fázisban melyik a legrosszabb eset, amely előfordulhat. Ezzel kapunk egy szignifikáns z vektort, amelyből egyrészt számolhatunk egy felső korlátot az optimumra (rögzített y mellett optimalizálunk), másrészt új változókat és a z értéke mellett rájuk vonatkozó korlátokat felvéve a mesterproblémához a következő iterációban jobb alsó korlátot kapunk. Az algoritmus akkor áll meg, amikor a két korlát összetalálkozik.
A modellünkre alkalmazva pedig a következőképpen néz ki.

1. Legyen az alsó korlát $LB = -\infty$, a felső korlát $UB = +\infty$, $n = 1$.

2. Oldjuk meg a mester problémát (MP):

$$\min (1-\varrho)\sum_i \sum_j c_{ij} d_i x_{ij} + \varrho\eta$$

$$x_{ij} \leq y_j \quad \forall i, j$$

$$\sum_i x_{ij} = 1 \quad \forall i$$

$$\sum_j y_j = p$$

$$x_{ij} \geq 0 \quad \forall i, j, \quad y_j \in \{0,1\} \quad \forall j$$

$$\eta \geq \sum_i \sum_j c_{ij}(1-\theta z_i^l) d_i w_{ij}^l + \sum_i M(1-\theta z_i^l) d_i q_i^l \quad \forall l = 1, .., n-1$$

$$\sum_j w_{ij}^l + q_i^l = 1 \quad \forall i, \ \forall l = 1, .., n-1$$

$$w_{ij}^l \leq 1 - z_j^l \quad \forall i, j, \ \forall l = 1, .., n-1$$

$$w_{ij}^l \leq y_j \quad \forall i, j, \ \forall l = 1, .., n-1$$

$$w_{ij}^l \geq 0 \quad \forall i, j, \ \forall l = 1, .., n-1, \quad q_i^l \geq 0 \quad \forall i, \ \forall l = 1, .., n-1, \quad \eta \geq 0$$

3. Állítsuk be LB-t a kapott optimumra, és legyen (y^n, x^n) MP optimális megoldása, ezt használva oldjuk meg a következő részproblémát (SP):

$$\max_z \min_{w,q} \sum_i \sum_j c_{ij}(1-\theta z_i) d_i w_{ij} + \sum_i M(1-\theta z_i) d_i q_i$$

$$w_{ij} \leq 1 - z_j \quad \forall i, j$$

$$w_{ij} \leq y_j^n \quad \forall i,j$$

$$\sum_j w_{ij} + q_i = 1$$

$$\sum_j z_j \leq k$$

$$w_{ij} \geq 0 \quad \forall i,j, \quad q_i \geq 0 \quad \forall i, \quad z_j \in \{0,1\} \quad \forall j$$

4. Legyen SP optimális megoldásának értéke $\mathcal{Q}(y^n)$, és az optimális z értéke pedig z^n.
   Legyen $UB = \min\{UB,\ (1-\rho)\sum_i \sum_j c_{ij} d_i x_{ij}^n + \rho \mathcal{Q}(y^n)\}$.

5. Ha $UB - LB \leq \epsilon$, akkor álljunk meg. Különben vegyünk MP-hez új (w^n, q^n) változókat a rájuk vonatkozó korlátokkal együtt z^n érték mellett. Legyen $n = n+1$, és menjünk a 2. lépéshez.

Ahhoz, hogy ez az algoritmus implementálható legyen, két kérdés merül fel SP-vel kapcsolatban. Az első az, amellyel az előző félévben már sokat foglalkoztunk, hogy van-e mindig megoldása, azaz $\mathcal{Q}(y^n)$ értéke véges-e. Ebben az esetben ugyanis nem lehetne egyszerűen lekérdezni a solvertől az optimális z^n értékét, vagyis azt a z vektort, amelyre nézve a feladat nem megengedett. Szerencsére itt SP mindig megengedett, még abban az esetben is az lenne, ha kapacitásos feladatot néznénk, hiszen lehetnek kiszolgálatlanul maradt igények, csak ezeket egy M értékkel büntetjük. Ez egyben azt is bizonyítja, hogy az algoritmus lépésszáma véges, hiszen legfeljebb annyi iteráció lesz, amennyi a $\{z \in \{0,1\}^{|J|} : \sum_j z_j \leq k\}$ halmaz elemeinek száma.
A másik kérdés az, hogy hogyan linearizáljuk SP max min alakú célfüggvényét, erre kétféle módszert próbáltam ki.
Az elsőben a belső minimalizálást hagyjuk el, és helyette felvesszük a probléma duális feltételeit (legyenek a duális változók u_{ij}, v_{ij}, s_i, amelyek rendre a (6)-(8) feltételekhez tartoznak), illetve a komplementaritási feltételeket. Ekkor a következő programot kapjuk.

$$\max_z \sum_i \sum_j c_{ij}(1-\theta z_i) d_i w_{ij} + \sum_i M(1-\theta z_i) d_i q_i$$

$$w_{ij} \leq 1 - z_j \quad \forall i,j$$

$$w_{ij} \leq y_j^n \quad \forall i,j$$

$$\sum_j w_{ij} + q_i = 1$$

$$u_{ij} + v_{ij} + s_i \leq c_{ij} d_i (1-\theta z_i) \quad \forall i,j$$

$$s_i \leq M d_i (1-\theta z_i) \quad \forall i$$

$$\sum_j z_j \leq k$$

$$w_{ij} > 0 \implies u_{ij} + v_{ij} + s_i = c_{ij} d_i (1-\theta z_i) \quad \forall i,j$$

$$q_i > 0 \implies s_i = M d_i (1-\theta z_i) \quad \forall i$$

$$u_{ij} < 0 \implies w_{ij} = 1 - z_j \quad \forall i,j$$

$$v_{ij} < 0 \implies w_{ij} = y_j^n \quad \forall i,j$$

$$w_{ij} \geq 0 \quad \forall i,j, \quad q_i \geq 0 \quad \forall i, \quad u_{ij} \leq 0 \quad \forall i,j, \quad v_{ij} \leq 0 \quad \forall i,j, \quad z_j \in \{0,1\} \quad \forall j$$

A második átírás [1]-ből származik, és az előzőhöz hasonlóan itt is a duális feltételeket vesszük fel, de a komplementaritási feltételek helyett a célfüggvényt írjuk át, és összeolvasztjuk a z-re vett maximumot és a duális problémában szereplő

maximalizálást. Az így kapott célfüggvény nemlineáris lesz, de a nagy M-módszer (amely egy megfelelően nagy $\mathbb{M}$ konstanst használ) segítségével linearizálhatjuk.

$$\max_{z,u,v,s} \sum_i \sum_j (1 - z_j) u_{ij} + \sum_i \sum_j y_j^n v_{ij} + \sum_i s_i$$

$$u_{ij} + v_{ij} + s_i \leq c_{ij} d_i (1 - \theta z_i) \quad \forall i, j$$

$$s_i \leq M d_i (1 - \theta z_i) \quad \forall i$$

$$\sum_j z_j \leq k$$

$$u_{ij} \leq 0 \quad \forall i, j, \quad v_{ij} \leq 0 \quad \forall i, j, \quad z_j \in \{0, 1\} \quad \forall j$$

A célfüggvényben az $u_{ij} z_j$ tag okozza a problémát (y_j^n értéke rögzített, így a második tag is lineáris), de bevezetve új $U_{ij} = u_{ij} z_j$ változókat, és kihasználva, hogy z bináris, a következőképpen küszöbölhetjük ezt ki.

$$\max_{z,u,v,s} \sum_i \sum_j (u_{ij} - U_{ij} + y_j^n v_{ij}) + \sum_i s_i$$

$$u_{ij} + v_{ij} + s_i \leq c_{ij} d_i (1 - \theta z_i) \quad \forall i, j$$

$$s_i \leq M d_i (1 - \theta z_i) \quad \forall i$$

$$U_{ij} \geq u_{ij} \quad \forall i, j$$

$$U_{ij} \geq -\mathbb{M} z_j \quad \forall i, j$$

$$U_{ij} \leq u_{ij} + \mathbb{M}(1 - z_j) \quad \forall i, j$$

$$\sum_j z_j \leq k$$

$$u_{ij} \leq 0 \quad \forall i, j, \quad U_{ij} \leq 0 \quad \forall i, j, \quad v_{ij} \leq 0 \quad \forall i, j, \quad z_j \in \{0, 1\} \quad \forall j$$

Mindkét módszert implementáltam, végül a második átírás bizonyult hatékonyabbnak, valószínűleg amiatt, hogy az első módszerben szereplő komplementaritási feltételek felírásához sok új bináris változót kell bevezetni, és ezek lelassítják a programot.

3. Futásidők és paraméterek

Az algoritmust részletesen teszteltem, hogy megvizsgálhassam, mely paraméterek hogyan hatnak a futásidőre. A tesztelést egy Asus ZenBook Pro 15-n (Intel Core i7-8750H, 2.20GHz, 16GB RAM) végeztem Windows 10 környezetben, a maximális időkeret két óra volt, ha ennyi idő alatt nem futott le, akkor azt a táblázatokban T-vel jelzem. A tesztfájlokat a következő alfejezetben is szereplő szlovák városokat tartalmazó adathalmazból generáltam az I és J indexek véletlenszerű választásával, az igényeket minden kliens esetén 1-nek definiáltam, és M érték úgy választottam, hogy nagyobb legyen, mint $\max_{i,j} c_{ij}$, és így mindig minden igény ki legyen szolgálva.

Két esetet vettem: amikor $I = J$ és amikor $J \subset I$. Először úgy választottam $|I|$ méretét, hogy a kétféle feladatban megegyezzen, de így olyan kis méretű feladatokat kaptam a $J \subset I$ esetben, hogy lényegében minden inputra egy másodpercen és két iteráción belül lefutott. Emiatt ebben az esetben az $|I| = 75$ mérettől indultam, míg az $I = J$ esetben ez volt a maximális méret, amit kipróbáltam. Tehát nem meglepő módon a feladat mérete hatással van a futásidőre.

Egy másik paraméter, amely látványosan befolyásolja, hogy mennyi idő alatt talál megoldást az algoritmus, az a modell

konzervatívságát szabályozó ϱ. Tapasztalataim szerint ennek függvényében monoton növekedik a feladat bonyolultsága, és így a futásidő.

A paraméterek közül még k volt az, amelynél összefüggést találtam a futásidővel. Igaz, hogy csak két értékére teszteltem le az algoritmust, de már ezekből is látszik, hogy k növekedésével nehezedik a feladat: az algoritmus által tett lépések száma egyenesen arányos a bizonytalan halmaz számosságával. Márpedig k növelésével ez a számosság nő (legalábbis $k = \frac{|J|}{2}$-ig, utána ismét csökken).

A paraméterek közül p és θ maradt ki. Az előbbit a feladat méretéhez igazítottam, így ezt nem elemzem külön, az utóbbinál pedig nem találtam összefüggést.

$\|I\| \times \|J\|$	p	ϱ	k	θ	Futásidő (mp)	Iterációk
25x25	5	0.1	1	-1	1.49	5
				0	2.534	6
				1	0.579	4
			2	-1	1.077	5
				0	15.063	13
				1	1.484	6
		0.5	1	-1	21.475	13
				0	23.186	13
				1	20.948	13
			2	-1	309.122	32
				0	4991.652	82
				1	374.365	38
50x50	10	0.1	1	-1	3.348	4
				0	4.509	5
				1	0.746	2
			2	-1	39.016	13
				0	81.567	14
				1	6.157	6
		0.5	1	-1	129.93	15
				0	101.231	14
				1	8.658	6
			2	-1	T	-
				0	T	-
				1	T	-
75x75	15	0.1	1	-1	10.542	5
				0	4.677	3
				1	13.427	5
			2	-1	48.864	8
				0	41.005	8
				1	71.962	10
		0.5	1	-1	520.023	15
				0	419.797	16
				1	239.112	13
			2	-1	T	-
				0	T	-
				1	T	-

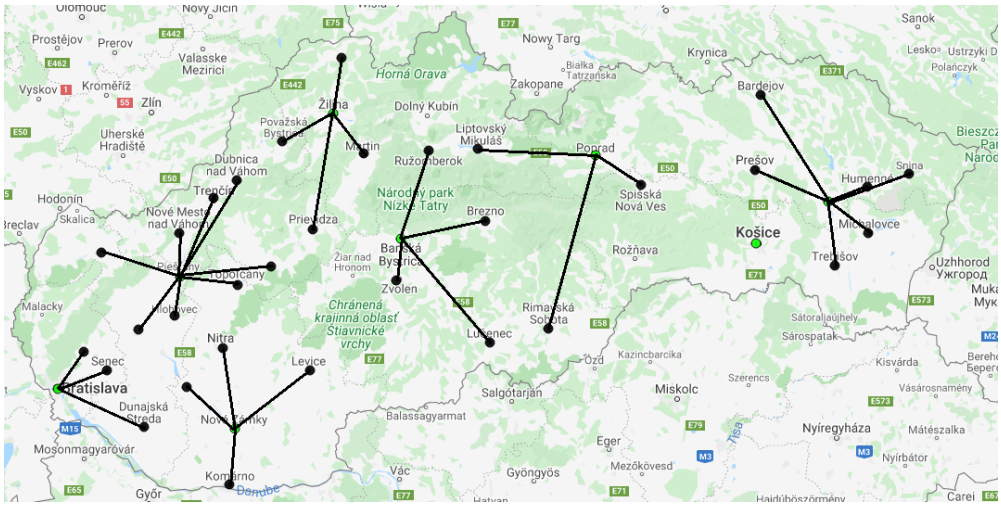
1. táblázat. Futásidők az $I = J$ esetben

$\|I\| \times \|J\|$	p	ϱ	k	θ	Futásidő (mp)	Iterációk
75x20	4	0.1	1	-1	0.547	2
				0	0.416	2
				1	0.453	2
			2	-1	1.081	3
				0	0.804	3
				1	0.86	3
		0.5	1	-1	3.059	5
				0	1.579	4
				1	1.785	4
			2	-1	12.538	7
				0	13.024	8
				1	10.322	6
100x25	5	0.1	1	-1	1.851	3
				0	1.595	3
				1	1.685	3
			2	-1	1.966	3
				0	2.717	4
				1	1.356	4
		0.5	1	-1	1.899	3
				0	1.788	3
				1	6.236	5
			2	-1	18.923	8
				0	15.662	7
				1	17.646	8
125x30	6	0.1	1	-1	1.163	2
				0	1.148	3
				1	0.908	2
			2	-1	6.022	5
				0	5.78	5
				1	5.157	6
		0.5	1	-1	2.621	3
				0	3.035	3
				1	5.462	4
			2	-1	48.561	10
				0	39.255	9
				1	32.65	9

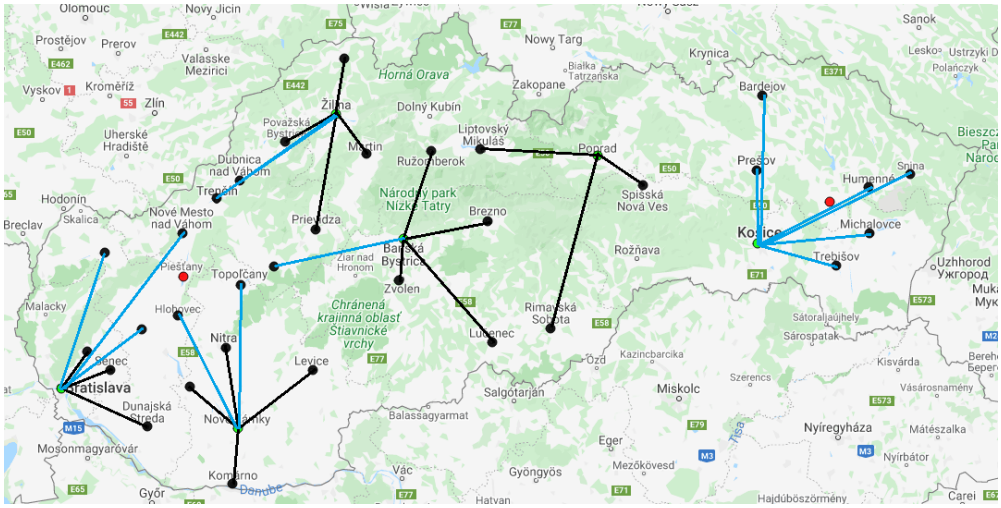
2. táblázat. Futásidők az $J \subset I$ esetben

4. Egy való életbeli példa

Az algoritmust kipróbáltam egy teljes egészében valós adatokon alapuló feladaton is. Ehhez a c_{ij} értékeket tartalmazó mátrixra volt szükségem, illetve a d_i igényvektorra. A távolságmátrixot a `http://frdsa.uniza.sk/~buzna/page5/supplement4/benchmarks.html` linkről vettem, ez egy 2928 szlovák települést és távolságaikat tartalmazó fájlcsomag. Ez olyan nagy feladat lenne, amely előreláthatólag nagyon lassan futna le, és a kapott eredményeket nehéz lenne ábrázolni is. Emiatt a feladat méretét jelentősen lecsökkentettem, és csak Szlovákia 40 legnagyobb városát vettem, határként a 20 ezres lélekszámot húztam meg. Az igényeket pedig a lakosságszám adta.
A következő paramétereket választottam: $|I| = |J| = 40$, $p = 8$, $\varrho = 0.25$, $k = 2$ és $\theta = 1$.



1. ábra. Normál működés a megadott paraméterek mellett



2. ábra. k=2 gyár leállása után

Az 1-es ábrán láthatóak az első fázisbeli normál működés melletti hozzárendelések, a térképeken zöld pöttyel jelöltem a megnyitott gyárak városait. Két érdekességet vehetünk észre, amelyek az ország szerkezetéből adódnak. Mivel a lakosságszámot használtam igényként, így nyilvánvaló volt, hogy Pozsonyban (Bratislava) fog nyitni gyárat az algoritmus. De mivel az ország délnyugati csücskében helyezkedik el, így kevés várost rendelt hozzá. A másik ehhez hasonló eset Kassa (Košice) város esete, amelyet szintén úgy nyitott meg az algoritmus, hogy kevés más várost szolgál ki, egész pontosan

egyet sem.
A 2-es ábrán látható, hogy a legrosszabb eset az, amikor az a két gyár áll le, amelyhez a legtöbb város volt rendelve, nyugaton Pöstyén (Piešťany) és keleten Varannó (Vranov nad Topľou). Ezeket az ábrán piros pötty jelzi, az újrakonfigurálást pedig a kék élek.

Hivatkozások

[1] Y. An, B. Zeng, Y. Zhang, and L. Zhao. Reliable *p*-median facility location problem: two-stage robust models and algorithms. *Transportation Research Part B*, 64:54–72, 2014.