Cjelobrojno programiranje: Problemi protoka. Problem maksimalnog protoka.

Ema Djedović
Prirodno-matematički fakultet
Univerzitet u Sarajevu
Odsjek za matematičke i kompjuterske nauke

03/2024

Sadržaj

Predgovor

Problemi s kojima se susrećemo u operacionim istraživanjima se često mogu modelirati u vidu neke mreže. Takvi mrežni problemi su primjenljivi u mnogim kontekstima koje čak ni linearno programiranje ne može dokučiti. Drugi su pak efikasno rješivi simpleks algoritmom nad mrežom (i to uglavnom nekom varijacijom specijaliziranom za konkretan problem), a neki su suviše jednostavni da bi se na njih primjenjivale tehnike linearnog programiranja.

U ovom izlaganju ćemo se pozabaviti mrežnim problemima protoka. Problem protoka minimalnog troška je onaj koji enkapsulira mnoge probleme s kojima smo se ranije mogli susresti, kao što su problem najkraćeg puta, problem transporta i problem asignacije. Uz navedene, tu je i problem maksimalnog protoka, kojim ćemo se više pozabaviti, a svi oni predstavljaju specijalne slučajeve problema protoka minimalnog troška.

U mnogim praktičnim problemima, jedino je smisleno da naše varijable imaju cjelobrojne vrijednosti (ili vrijednosti iz bilo kojeg diskretnog skupa). Generalno, probleme linearnog programiranja bez ovog ograničenja je mnogo lakše rješavati. Iz tog razloga cjelobrojnim problemima uglavnom pridružujemo odgovarajući linearni problem, odnosno vršimo LP relaksaciju. Ukoliko smo dovoljne sreće da optimalno rješenje linearnog problema zadovoljava uslove cjelobrojnosti varijabli, onda smo time dobili i optimalno rješenje cjelobrojnog programa. Naći se u ovakvoj poziciji je doista rijetkost - osim ako ne radimo sa specijalnim tipovima cjelobrojnih problema za koje je ovaj ishod garantiran. Problem protoka minimalnog troška (s cjelobrojnim parametrima) je jedan od njih. Ova činjenica sa sobom povlači da su i problem najkraćeg puta, problem transporta, asignacije i problem maksimalnog protoka kao specijalni slučajevi također rješivi na ovaj način.

Mrežni problemi jednostavno vape za grafovskom reprezentacijom, što će biti očito kroz primjere. Vrijedi i obrnuto, veliki broj problema teorije grafova može se prikazati preko modela matematičkog programiranja.

1 Problem protoka minimalnog troška

Uzmimo u obzir usmjereni i povezani graf u kojem n čvorova uključuje barem jedan izvor i barem jedan utok. Varijable odluke su:

x_{ij} = protok kroz granu (i, j),

a dati podaci uključuju

c_{ij} = trošak po jedinici protoka kroz granu (i, j),
u_{ij} = kapacitet za granu (i, j),
b_i = neto protok generiran na čvoru i

Vrijednost b_i ovisi o prirodi čvora i, gdje

$b_i < 0$ ako je čvor i izvor,
$b_i > 0$ ako je čvor i utok,
$b_i = 0$ ako je čvor i neutralni čvor.

Cilj je minimizirati ukupni trošak slanja robe kroz mrežu kako bi se zadovoljila potražnja na utocima.

$$\begin{aligned} \text{Minimizirati} \quad & Z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \text{uz ograničenja} \quad & \sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{ji} = b_i, \quad \text{za svaki čvor } i \\ & 0 \le x_{ij} \le u_{ij}, \quad \text{za svaku granu } (i,j) \end{aligned}$$

Prva suma u ograničenjima čvorova predstavlja ukupni protok iz čvora i, dok druga suma predstavlja ukupni protok u čvor i, pa je razlika neto protok generiran na ovom čvoru. Navedena ograničenja nazivaju se jednadžbama očuvanja protoka (bilanca čvorova, Kirchhoffove jednadžbe) i ukazuju da se protok ne smije stvarati niti uništavati u mreži.

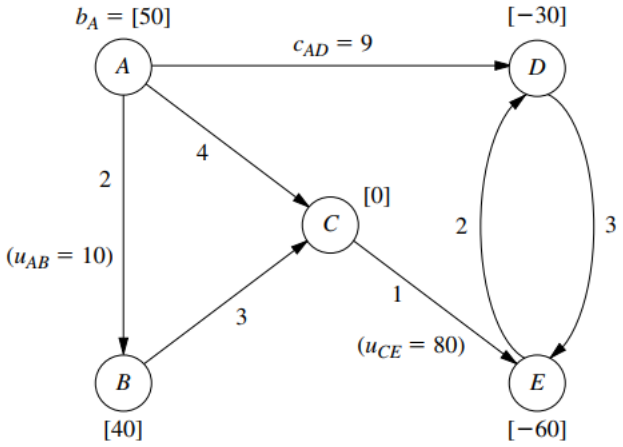
Ovako formulisan problem ne uključuje gornje granice na granama, zbog čega se također naziva i neograničenim. U nekim primjenama, potrebno je imati donju granicu $L_{ij} \ge 0$ za protok kroz svaki luk (i, j). Kada se to dogodi, koristimo translacije $x'_{ij} = x_{ij} - L_{ij}$, te kroz cijeli model koristimo $x'_{ij} + Lij$ umjesto x_{ij}.

Pretpostavit ćemo da ukupna ponuda odgovara ukupnoj potražnji, odnosno,

$\sum_{i=1}^{n} b_i = 0$. Ukoliko je pak $\sum_{i=1}^{n} b_i > 0$, tada možemo dodati fiktivni čvor utok za koji vrijedi $b_{n+1} = -\sum_{i=1}^{n} b_i$ i dodati grane s nultom cijenom od svakog izvora do ovog novog čvora. Analogno za $\sum_{i=1}^{n} b_i < 0$ dodajemo fiktivni čvor izvor i nove grane s nultom cijenom od njega do utoka.

Primjer

Predstavimo jedan primjer problema protoka minimalnog troška na distribucijskoj mreži. U uglastim zagradama [ ] su date količine b_i. Vidimo da su čvorovi izvori A i B ($b_i > 0$), a čvorovi utoci D i E ($b_i < 0$). Čvor C je neutralni čvor ($b_i = 0$). Date su cijene transporta c_{ij} po granama, kao i kapacitet protoka u_{ij} za dvije grane. Ostale grane nisu ograničene kapacitetom protoka ($u_{ij} = \infty$).



Linearni program je sljedeći:

Minimizirati $Z = 2x_{AB} + 4x_{AC} + 9x_{AD} + 3x_{BC} + x_{CE} + 3x_{DE} + 2x_{ED}$,
uz ograničenja:

$$\begin{aligned} x_{AB} + x_{AC} + x_{AD} &= 50 \\ -x_{AB} + x_{BC} &= 40 \\ -x_{AC} - x_{BC} + x_{CE} &= 0 \\ -x_{AD} + x_{DE} - x_{ED} &= -30 \\ -x_{CE} - x_{DE} + x_{ED} &= -60 \end{aligned}$$

gdje je $x_{AB} \leq 10$, $x_{CE} \leq 80$, za sve $x_{ij} \geq 0$.

Primjene

Problem protoka minimalnog troška zauzima centralno mjesto među modelima optimizacije mreže, kako zbog toga što obuhvaća tako širok spektar primjena, tako i zbog toga što se može riješiti izuzetno efikasno. Može se pojaviti u logističkoj mreži u kojoj se ljudi i materijali premještaju između različitih tačaka u svijetu. Može biti povezan s kretanjem lokomotiva između tačaka u željezničkoj mreži kako bi se zadovoljila potreba za vozovima minimizirajući troškove putovanja. Dalje, u dizajnu i analizi komunikacijskih sustava, problema lanca opskrbe i distribucije, sustava cjevovoda za naftu, problema rasporeda tankera itd.

2 Totalno-unimodularne matrice

Definicija 2.1 Za matricu A kažemo da je totalno-unimodularna ako je determinanta svake njene podmatrice +1, -1 ili 0.

Tvrdnja 2.1 *Matrica A je totalno-unimodularna ako:*

(i) A^T *je totalno-unimodularna,*

(ii) (A, I) *je totalno-unimodularna.*

Opažanje 2.1 Ako je matrica A totalno-unimodularna, onda je $a_{ij} \in \{+1, -1, 0\}$ za svako i, j.

Obrat nužno ne vrijedi. Drugim riječima, postojanje ovih vrijednosti ne garantuje totalnu-unimodularnost matrice. Pogledajmo dva primjera.

Primjer 2.1 Matrica P je totalno-unimodularna, dok matrica Q nije.

$$P = \begin{pmatrix} 1 & -1 & -1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad Q = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

Na osnovu sljedeće tvrdnje možemo jasno vidjeti zašto ovo vrijedi.

Tvrdnja 2.2 *Matrica A je totalno-unimodularna ako:*

(i) $a_{ij} \in \{+1, -1, 0\}$ *za sve* i, j,

(ii) Svaka kolona matrice A sadrži najviše dva nenulta elementa, odnosno $(\sum_{i=1}^{m} \|a_{ij}\| \leq 2)$,

(iii) Postoji particija $(R1, R2)$ *skupa redova R takva da svaka kolona j koja sadrži dva nenulta elementa zadovoljava uslov*

$$\sum_{i \in R1} a_{ij} - \sum_{i \in R2} a_{ij} = 0$$

.

Veza s cjelobrojnošću rješenja linearnih programa

Prirodno je da prvi korak u rješavanju problema cjelobrojnog programiranja oblika

$$\min/\max \{cx : Ax \leq b, x \in \mathbb{Z}_+^n\}$$

bude postavljanje pitanja: Hoćemo li imati sreće da njemu pridruženi linearni program pruži optimalno rješenje koje je cjelobrojno? Drugim riječima, bilo bi zgodno kada bi LP relaksacija

$$\min/\max \{cx : Ax \leq b, x \in \mathbb{R}_+^n\}$$

imala cjelobrojno rješenje.

Iz teorije linearnog programiranja, znamo da su bazična moguća rješenja oblika $x = (x_B, x_N) = (B^{-1}b, 0)$ gdje je B invertibilna $m \times m$ podmatrica od (A, I). I je jedinična matrica dimenzije $m \times m$.

Opažanje 2.2 Ako optimalna baza B ima determinantu $\det(B) = \pm 1$, onda linearna (LP) relaksacija rješava odgovarajući cjelobrojni problem (IP).

Dokaz Kramerovo pravilo nam govori da je $B^{-1} = B^*/\det(B)$, gdje je B^* adjungovana matrica. Kako su elementi matrice B^* proizvodi elemenata iz B, tako je B^* cjelobrojna matrica. Iz $\det(B) = \pm 1$ zaključujemo da je B^{-1} također cjelobrojna. Stoga, B^{-1} je cjelobrojna za sve cjelobrojne vektore b.

Sljedeći korak je razmotriti u kojim slučajevima ćemo imati da sve (optimalne) baze zadovoljavaju $\det(B) = \pm 1$?

Tvrdnja 2.3 *Linearni program* $\max\{cx : Ax \leq b, x \in \mathbb{R}_+^n\}$ *ima optimalno cjelobrojno rješenje za sve cjelobrojne vektore* b *ako i samo ako je matrica* A *totalno-unimodularna.*

3 Cjelobrojnost u problemu protoka

U prethodnom poglavlju smo predstavili jednostavan algoritam kojim ćemo provjeriti totalnu-unimodularnost matrice.

Osmotrimo ponovo našu formulaciju problema protoka:

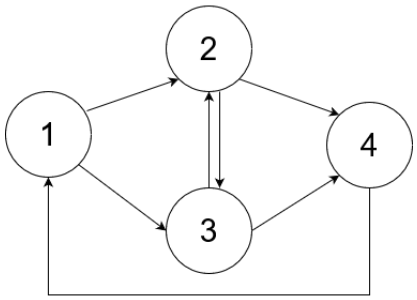
$$\begin{aligned} \min \quad & Z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\ & \sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{ji} = b_i, \quad \text{za svaki čvor } i \\ & 0 \leq x_{ij} \leq u_{ij}, \quad \text{za svaku granu } (i,j) \end{aligned}$$

Matrica ograničenja A ima jedan red za svaki čvor grafa i jednu kolonu za svaku granu. Svaka kolona matrice A sadrži tačno dva nenultna koeficijenta: "+1" i "-1". Kolona povezana s granom (i, j) sadrži "+1" u redu i, "-1" u redu j, i nule drugdje. Dakle, elementi matrice A dati su sljedećim izrazom:

$$a_{ij} = \begin{cases} e_i - e_j, & \text{ako postoji grana od čvora } i \text{ do čvora } j, \\ 0, & \text{inače.} \end{cases}$$

gdje su e_i i e_j jedinični vektori u $\mathbb{R}^m$, s jedinicama na i-tom i j-tom mjestu, redom. Matrica A naziva se matricom čvor-grana za dati graf.

Dokažimo kroz primjer da je matrica A totalno-unimodularna. Posmatrajmo graf na slici i njemu pridruženu matricu ograničenja A.



Slika 1: Usmjereni graf s četiri čvora i sedam grana.

$$\mathbf{A} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccccc} (1,2) & (1,3) & (2,3) & (2,4) & (3,2) & (3,4) & (4,1) \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 1 & 1 & -1 & 0 & 0 \\ 0 & -1 & -1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & -1 & 1 \end{bmatrix} \end{array}$$

Slika 2: Pridružena matrica ograničenja.

Koristit ćemo princip matematičke indukcije. Prije svega vidimo da, kako su svi elementi matrice iz skupa $\{+1, -1, 0\}$ tako je determinanta svake 1×1 podmatrice jednaka ± 1 ili 0. Ovo je baza indukcije.

Dalje, pretpostavimo da tvrdnja važi za sve kvadratne podmatrice dimenzije $(k-1) \times (k-1)$ i neka je A_k proizvoljna $k \times k$ podmatrica od A, $k \geq 2$. Želimo pokazati da vrijedi $\det(A_k) = \pm 1$ ili 0.

Uočimo da svaka kolona od A_k može imati sve nule, samo jedan nenulti element ($+1$ ili -1), ili tačno dva nenulta elementa ($+1$ i -1). Ako je bilo koja kolona od A_k nulta, onda je $\det(A_k) = 0$. Ako bilo koja kolona od A_k ima jedan nenulti element, onda proširivanjem determinante od A_k minorima te kolone dobivamo $\det(A_k) = \pm \det(A_{k-1})$, gdje je A_{k-1} kvadratna podmatrica dimenzije $(k-1) \times (k-1)$. Induktivnom hipotezom $\det(A_{k-1}) = \pm 1$ ili 0 zaključujemo da vrijedi $\det(A_k) = \pm 1$ ili 0. Ukoliko svaka kolona od A_k ima oba elementa $+1$ i -1, sabiranjem kolona od A_k dobivamo nul-vektor, pa je $\det(A_k) = 0$. Dakle, A je totalno-unimodularna.

Uočimo da rečeno stoji čak i ukoliko brišemo redove ili kolone iz A, ili (još važnije) ukoliko dodajemo jedinične vektor-kolone u matricu A. Ove kolone mogu predstavljati recimo fiktivne varijable. Rezultirajuća matrica ostaje totalno-unimodularna.

Šta ovo znači? Znači da ćemo problemu protoka uvijek moći pridružiti LP relaksaciju čijim ćemo rješavanjem dobiti *cjelobrojna optimalna rješenja.*

Zaključak

Kada su svi parametri b_i i u_{ij} cjelobrojni, svojstvo totalne unimodularnosti matrice A osigurava da sve varijable u optimalnom rješenju također imaju cjelobrojne vrijednosti. To znači da će rješenja problema protoka minimalnog troška biti cjelobrojna, čak i ako nisu eksplicitno postavljene cjelobrojne granice na varijable.

4 Specijalni slučajevi

Problem najkraćeg puta

U ovom problemu nam je predstavljen neusmjeren graf. Da bismo preformulisali problem u obliku protoka minimalnog troška zamijenit ćemo svaku granu parom usmjerenih grana u suprotnim smjerovima. Tražimo najkraći put od početnog čvora (jedan izvor) do krajnjeg čvora (jedan utok). Tim čvorovima se pridružuje količina 1. Ostali čvorovi su neutralni čvorovi količine 0. Udaljenost između čvorova i i j postaje jedinični trošak c_{ij} ili c_{ji} za protok u bilo kojem smjeru između tih čvorova. Ne postavljaju se ograničenja kapaciteta grana, pa su svi $u_{ij} = \infty$.

Problem transporta

Trebamo prevesti robu od čvorova izvora do čvorova utoka, po potrebi koristeći neutralne čvorove. Distribuiranje x_{ij} jedinica robe od izvora i do odredišta j odgovara protoku x_{ij} kroz granu (i, j). Budući da problem transporta ne nameće gornje granice na pojedinačne x_{ij}, svi kapaciteti su $u_{ij} = \infty$. Ovaj problem se često susreće u logistici, gdje je potrebno efikasno rasporediti robu između različitih lokacija, uz minimalne troškove prijevoza.

Problem asignacije

Ovaj problem se može posmatrati kao posebna vrsta problema transporta, uz uslove da je broj čvorova izvora jednak broju čvorova utoka, te da je $b_i = 1$ za svaki čvor izvor i $b_i = -1$ za svaki čvor utok. Asignacija se često koristi u ekonomiji i operacijskim istraživanjima za raspoređivanje resursa na način koji optimizira ukupne koristi ili minimizira ukupne troškove.

Problem maksimalnog protoka

Dat je usmjeren graf koji predstavlja protok u mreži tako da je grani (i, j) pridružen kapacitet u_{ij}. Dati su čvor izvor s i čvor utok t. Potrebno je naći maksimalni mogući protok od izvora s do utoka t tako da vrijedi:

a) Protok kroz bilo koju granu ne prelazi dati kapacitet za tu granu

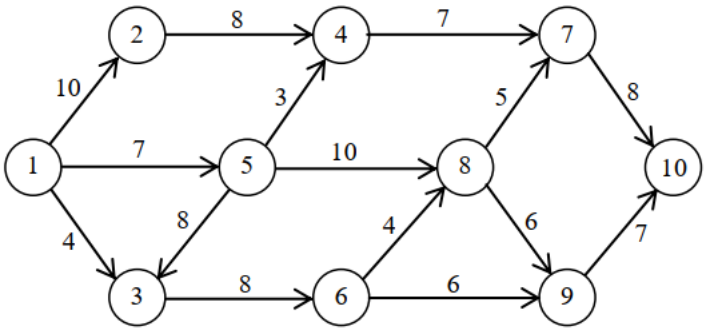
b) Količina koja ulazi u čvor je jednaka količini koja izlazi kroz isti, za sve čvovore sem izvora i utoka.

Primjene se najčešće susreću u raznim tipovima realnih transportnih zadataka. Recimo, ako je iz jednog grada za određeno vrijeme potrebno prebaciti što više putnika u drugi grad mrežom avionskog saobraćaja, dobija se upravo zadatak maksimalnog protoka. Podrazumijeva se da svaka relacija avionskog saobraćaja ima svoju propusnu sposobnost, tj. gornju granicu "tereta" koji se može po njoj prevesti za određeno vrijeme. Pri tome se pretpostavlja da se putnici ne mogu ni ukrcavati ni iskrcavati u međustanicama, inače se zadatak usložnjava.

Osim u transportnim problemima, maksimalni protok se često koristi i u modeliranju komunikacijskih mreža kako bi se osiguralo optimalno iskorištavanje resursa i minimizirale gužve.

5 Problem maksimalnog protoka

Neka je n broj čvorova grafa. Bez umanjenja opštosti, uzmimo da je čvor izvor označen sa 1, a čvor utok sa n. Varijable x_{ij} predstavljaju protoke kroz grane od čvora i do čvora j. Kapacitete grana označimo sa u_{ij}.



Slika 3: Primjer mreže s 10 čvorova, gdje su čvor 1 izvor i čvor 10 utok.

Ukoliko sa F označimo ukupni protok kroz mrežu, tada se zadatak nalaženja maksimalnog protoka može zapisati u obliku:

$$\max F = \sum_{j=2}^{n} x_{1j}$$

$$\sum_{i=1}^{n} x_{ik} - \sum_{j=1}^{n} x_{kj} = 0, \quad k = 2..n-1$$

$$0 \leq x_{ij} \leq u_{ij}, \quad i = 1..n, \quad j = 1..n$$

Prva skupina ograničenja predstavlja jednačine očuvanja toka za sve čvorove osim izvora i utoka. One odražavaju činjenicu da u tim čvorovima ne nastaje niti nestaje tok, tako da ukupan protok koji ulazi u čvor mora biti jednak ukupnom protoku koji izlazi iz čvora. Druga skupina ograničenja predstavlja ograničenja maksimalnog toka kroz grane u skladu s njihovim kapacitetima. Ukoliko grana od i do j ne postoji, tada uzimamo x_{ij}=0 i u_{ij}=0.

S obzirom da spada u specijalan slučaj problema minimalnog troška, očito je da problem maksimalnog protoka ima matricu ograničenja A koja je totalno unimodularna. Slijedi da, ako su kapaciteti svih grana cjelobrojni, tada uvijek imamo *cjelobrojna optimalna rješenja.*

5.1 Primjene

Problem maksimalnog protoka ima širok spektar primjena u različitim područjima, uključujući, ali ne ograničavajući se na sljedeće:

Telekomunikacijske mreže

Maksimalni protok se koristi za optimizaciju protoka podataka kroz mreže, kao što su internet mreže, mobilne mreže i optičke mreže. To uključuje usmjeravanje paketa podataka kroz rutere i prekidače kako bi se maksimizirao protok informacija. Kapaciteti grana predstavljaju maksimalnu propusnost veza između čvorova u mreži, obično izraženu u bitima po sekundi (bps) ili u drugim relevantnim jedinicama za prijenos podataka.

Transportni sistemi

Primjena maksimalnog protoka uključuje planiranje optimalnih ruta u cestovnom prometu, zračnom prometu, željezničkom prometu i pomorskom prometu kako bi se maksimizirao protok putnika ili tereta. Kapaciteti grana predstavljaju maksimalni kapacitet prometa koji može proći kroz svaku cestu, željezničku liniju, zračnu rutu ili morski put. To bi moglo biti izraženo u broju vozila po satu ili drugim relevantnim jedinicama za promet.

Logistika i distribucija

U distribucijskim mrežama, kao što su lanac opskrbe ili distribucijski centri, maksimalni protok se koristi za optimizaciju protoka robe ili materijala između različitih tačaka kako bi se minimizirali troškovi ili vrijeme isporuke. Kapaciteti grana odražavaju maksimalni kapacitet transportnih sredstava ili skladišnih prostora na svakoj ruti ili u svakom skladištu, izraženo u volumenu ili težini robe koja može proći kroz svaku tačku.

Proizvodnja i operacije

U proizvodnim procesima, problemi maksimalnog protoka se koriste za optimizaciju protoka materijala ili proizvoda kroz različite faze proizvodnje kako bi se maksimizirala produktivnost i iskoristivost resursa. Kapaciteti grana u proizvodnim mrežama mogu predstavljati maksimalni kapacitet proizvodnih linija ili resursa (poput strojeva ili radne snage) koji mogu obraditi određenu količinu materijala ili proizvoda u jedinici vremena.

Energetske mreže

U električnim mrežama, maksimalni protok se koristi za optimalno usmjeravanje energije između različitih izvora i potrošača kako bi se maksimizirala učinkovitost i pouzdanost distribucije električne energije. Kapaciteti grana u energetskim mrežama odražavaju maksimalnu snagu koja se može prenositi kroz električne žice ili druge komponente mreže, obično izraženu u kilovatima (kW) ili megavatima (MW).

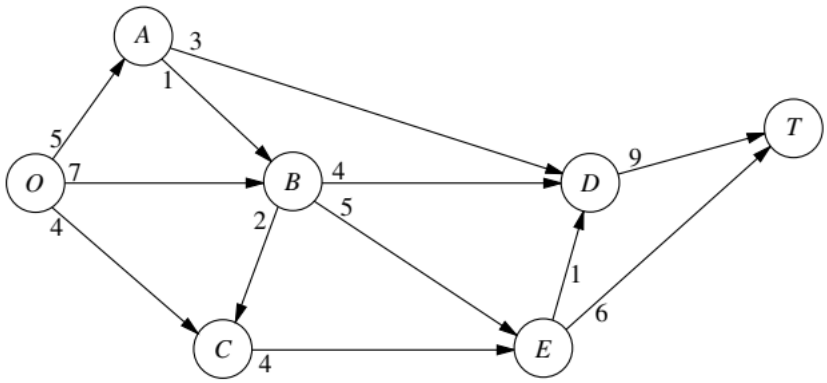
Biološki sistemi

Primjena maksimalnog protoka u biološkim sistemima uključuje analizu protoka krvi kroz krvne žile ili protoka signala u neuronskim mrežama radi boljeg razumijevanja bioloških procesa i funkcija. U biološkim mrežama, kapaciteti grana mogu predstavljati maksimalni protok krvi kroz krvne žile ili maksimalni protok signala kroz neuronske veze, što može biti implicitno određeno fiziološkim ili biokemijskim karakteristikama tih sistema.

Za neke od ovih primjena, protok kroz mrežu može potjecati iz više čvorova i također se može završiti na više čvorova, iako je u problemu maksimalnog protoka dopušteno imati samo jedan izvor i jedan utok. Na primjer, distribucijska mreža tvrtke obično ima više tvornica i više kupaca. Koristi se pametna reformulacija kako bi se takva situacija prilagodila problemu maksimalnog protoka. Ova reformulacija uključuje proširenje originalne mreže kako bi se uključili fiktivni izvor, fiktivni utok i neke nove veze. Fiktivni izvor se tretira kao čvor iz kojeg potječe sav protok koji, u stvarnosti, potječe iz nekih drugih čvorova. Za svaki od tih drugih čvorova, ubacuje se nova veza koja vodi od fiktivnog izvora do tog čvora, pri čemu kapacitet te veze odgovara maksimalnom protoku koji, u stvarnosti, može potjecati iz tog čvora. Slično tome, fiktivni utok se tretira kao čvor koji upija sav protok koji, u stvarnosti, završava na nekim drugim čvorovima. Stoga se iz svakog od tih drugih čvorova umeće nova veza do fiktivnog utoka, pri čemu kapacitet te veze odgovara maksimalnom protoku koji, u stvarnosti, može završiti na tom čvoru.

5.2 Algoritam

Osmotrimo usmjeren graf sa slike 4. Čvor izvor je označen sa O, čvor utok sa T, a usmjerenim granama su pridruženi kapaciteti protoka.



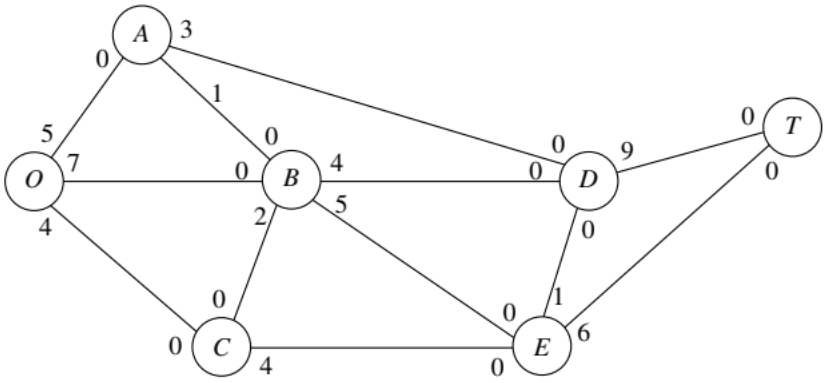
Slika 4: Inicijalni problem predstavljen preko usmjerenog grafa sa kapacitetima protoka za svaku granu.

Algoritam za rješavanje problema maksimalnog protoka se temelji na dva intuitivna koncepta, rezidualnoj mreži i povećavajućoj putanji. Nakon što su neki protoci dodijeljeni granama, **rezidualna mreža** prikazuje preostale kapacitete grana (zovemo ih **rezidualni kapaciteti**) za dodjelu budućih protoka. Na primjer, razmotrimo granu $O \rightarrow B$ na slici 4, koja ima kapacitet 7. Sada pretpostavimo da dodijeljeni protoci uključuju protok od 5 kroz ovu granu, što ostavlja rezidualni kapacitet od 7 - 5 = 2 za buduće dodjele protoka kroz $O \rightarrow B$. U rezidualnoj mreži ovaj koncept je prikazan na sljedeći način:

Slika 5: Rezidualni kapaciteti u oba smjera.

Broj na grani pored čvora daje rezidualni kapacitet za protok iz tog čvora u drugi čvor. Stoga, osim rezidualnog kapaciteta od 2 za protok iz O u B, broj 5 s desne strane ukazuje na rezidualni kapacitet od 5 za dodjelu nekog protoka iz B u O (tj. za poništavanje nekog prethodno dodijeljenog protoka iz O u B).

Na početku, prije nego što su dodijeljeni bilo kakvi protoci, rezidualna mreža ima izgled prikazan na slici 6. Svaka grana u originalnoj mreži (slika 4) promijenjena je iz usmjerene u neusmjerenu granu. Međutim, kapacitet grane u originalnom smjeru ostaje isti, a kapacitet grane u suprotnom smjeru je nula, pa su ograničenja na protok nepromijenjena.



Slika 6: Pripadajući neusmjereni graf s kapacitetima grana u oba smjera.

Nakon toga, svaki put kada se odredi određena količina protoka kroz granu, ta količina se oduzima od rezidualnog kapaciteta u istom smjeru i dodaje se rezidualnom kapacitetu u suprotnom smjeru.

Povećavajuća putanja je usmjerena putanja od izvora do utoka u rezidualnoj mreži takva da svaka grana na ovoj putanji ima strogo pozitivan rezidualni kapacitet. Minimum ovih rezidualnih kapaciteta naziva se **rezidualni kapacitet povećavajuće putanje** jer predstavlja količinu protoka koja se može dodati na cijelu putanju. Stoga, svaka povećavajuća putanja pruža mogućnost daljnjeg povećanja protoka kroz originalnu mrežu.

Algoritam povećavajućih putanja u više navrata odabire neku povećavajuću putanju i dodaje protok jednak njenom rezidualnom kapacitetu na tu putanju u originalnoj mreži. Taj proces se nastavlja sve dok više nema povećavajućih putanja, tako da protok od izvora do utoka ne može dalje biti povećan. Konačno rješenje mora biti optimalno zbog činjenice da povećavajuće putanje mogu poništiti neke prethodno dodijeljene protoke u originalnoj mreži, tako da nas loša dodjela protoka ne može spriječiti da kasnije dobijemo neku bolju alternativu.

Pretpostavljajući da su kapaciteti grana cijeli ili racionalni brojevi, svaka iteracija algoritma se sastoji od sljedeća tri koraka:

1. Identificiramo povećavajuću putanju pronalaženjem neke usmjerene putanje od izvora do utoka u rezidualnoj mreži tako da svaka grana na ovoj putanji ima strogo pozitivan rezidualni kapacitet. Ako ne postoji povećavajuća putanja tada protoci koje smo do tog trenutka dodijelili čine optimalno rješenje.

2. Identificiramo rezidualni kapacitet c^* izabrane povećavajuće putanje pronalaženjem minimuma rezidualnih kapaciteta grana na ovoj putanji. Povećavamo protok na ovoj putanji za c^*.

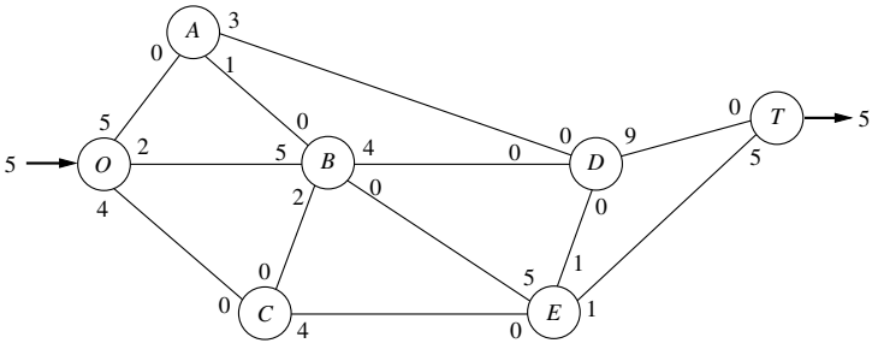
3. Sada ćemo na ovoj povećavajućoj putanji smanjiti rezidualni kapacitet svake grane za c^*, te za istu vrijednost c^* povećati rezidualni kapacitet svake grane u suprotnom smjeru. Vraćamo se na korak 1.

Nakon sprovedbe koraka 1, često će postojati nekoliko alternativnih povećavajućih putanja koje se mogu odabrati. Iako je algoritamska strategija za izbor važna za učinkovitost implementacija na mrežama velikog opsega, nećemo ulaziti u ovu relativno specijaliziranu temu. Stoga, za sljedeći primjer odabir povećavajuće putanje će se izvršavati proizvoljno.

5.3 Primjer

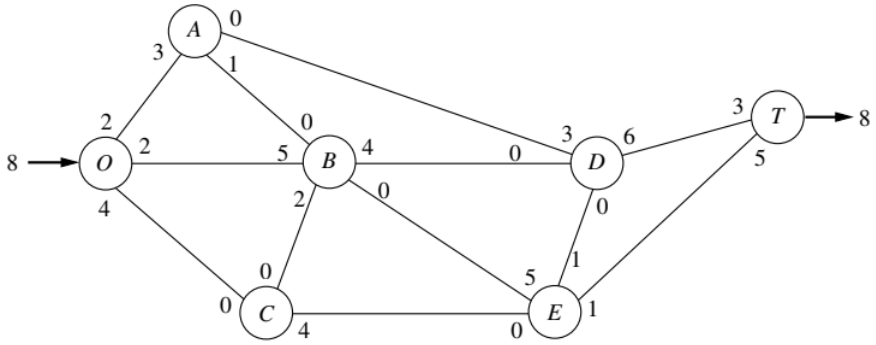
Počevši s rezidualnom mrežom prikazanom na slici 5, dajemo novu rezidualnu mrežu nakon svake iteracije, ili njih dvije, gdje je ukupna postignuta količina protoka od O do T prikazana podebljano (do čvorova O i T).

Iteracija 1: Na slici 6, jedna od nekoliko povećavajućih putanja je $O \to B \to E \to T$, koja ima rezidualni kapacitet $\min\{7, 5, 6\} = 5$. Dodjelom protoka od 5 ovoj putanji, ovo je rezultirajuća rezidualna mreža:



Slika 7: Nakon iteracije 1.

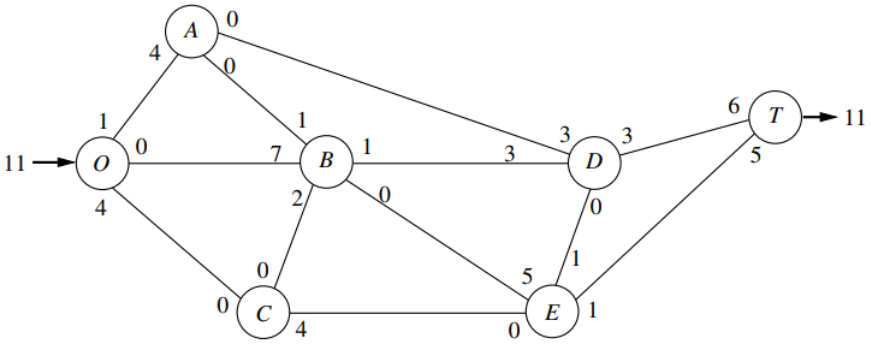
Iteracija 2: Dodijelit ćemo protok od 3 povećavajućoj putanji $O \to A \to D \to T$.



Slika 8: Nakon iteracije 2.

Iteracija 3: Dodijelit ćemo protok od 1 povećavajućoj putanji $O \to A \to B \to D \to T$.

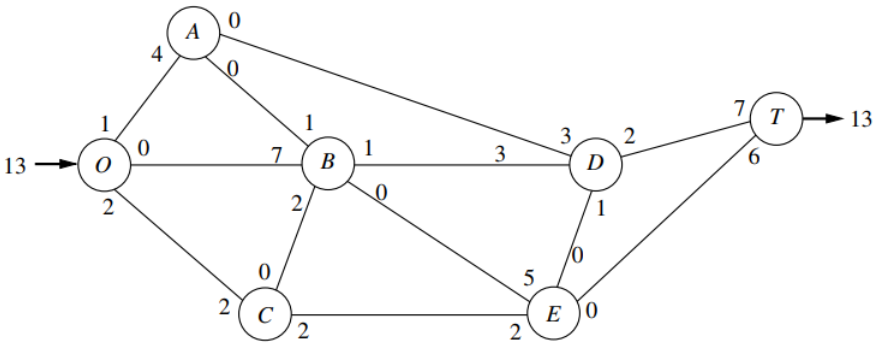
Iteracija 4: Dodijelit ćemo protok od 2 povećavajućoj putanji $O \to B \to D \to T$.



Slika 9: Nakon iteracija 3 i 4.

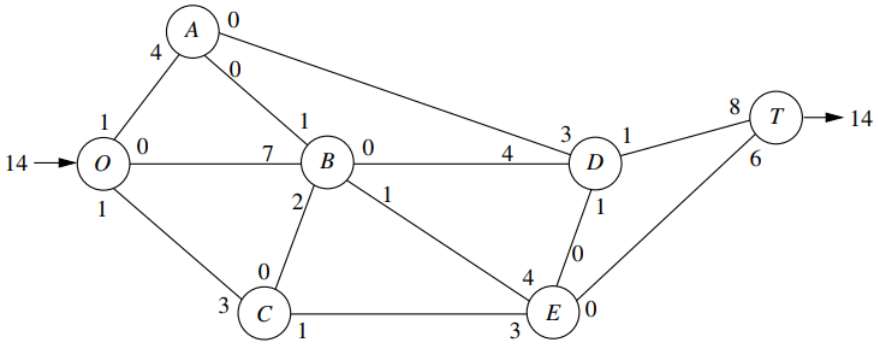
Iteracija 5: Dodijelit ćemo protok od 1 povećavajućoj putanji $O \to C \to E \to D \to T$.

Iteracija 6: Dodijelit ćemo protok od 1 povećavajućoj putanji $O \to C \to E \to T$.



Slika 10: Nakon iteracija 5 i 6.

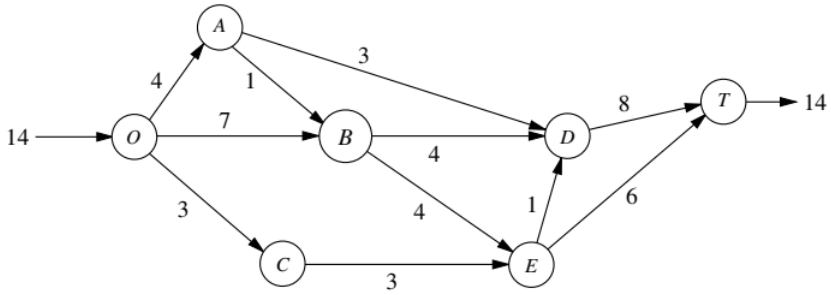
Iteracija 7: Dodijelit ćemo protok od 1 povećavajućoj putanji $O \to C \to E \to B \to D \to T$.



Slika 11: Nema više povećavajućih putanja.

Trenutna šema protoka može se identificirati ili zbrajanjem dodijeljenih protoka ili usporedbom krajnjih rezidualnih kapaciteta s izvornim kapacitetima grana. Ako koristimo potonju metodu, tada postoji protok duž neke grane ako je njen konačni rezidualni kapacitet manji od izvornog kapaciteta. Količina protoka jednaka je razlici između tih kapaciteta.

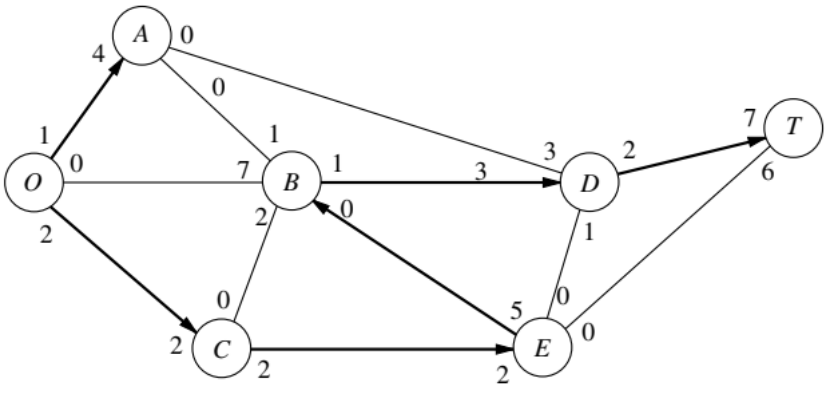
Primjenom ove metode usporedbe rezidualne mreže dobivamo optimalnu šemu protoka prikazanu na slici 12.



Slika 12: Optimalno rješenje za naš primjer.

5.4 Pronalaženje povećavajuće putanje

Kada radimo s velikim mrežama, najteži dio ovog algoritma je pronalaženje povećavajuće putanje. Ovaj zadatak može biti pojednostavljen sljedećim sistematičnim postupkom. Počinjemo određivanjem svih čvorova do kojih se može doći iz izvora duž jedne grane sa strogo pozitivnim rezidualnim kapacitetom. Zatim, za svaki od tih dosegnutih čvorova, odredit ćemo sve nove čvorove (one koji nisu još dosegnuti) do kojih se može doći iz ovog čvora duž grane ponovo sa strogo pozitivnim rezidualnim kapacitetom. Nastavljamo ovako s novim čvorovima redom kako ih dosežemo. Rezultat će biti stablo svih čvorova do kojih se može doći iz izvora duž putanje s strogo pozitivnim kapacitetom protoka. Tako će ovaj postupak uvijek identificirati povećavajuću putanju ukoliko ista postoji. Ukoliko krenemo iz iteracije 6 u prethodnom primjeru, onda imamo:



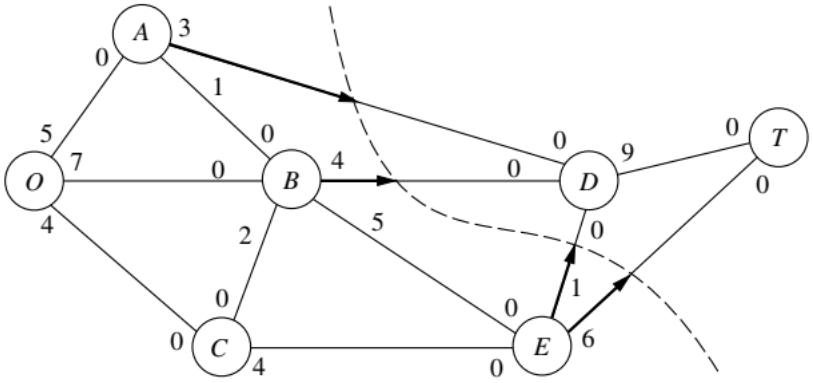
Slika 13: Postupak identifikacije povećavajuće putanje.

Iako je navedeni postupak relativno jednostavan, bilo bi korisno bez iscrpnog pretraživanja prepoznati kada smo dosegli optimalnost. Od koristi nam je tzv. *Teorem o maksimalnom protoku i minimalnom rezu.*

Rez se može definirati kao bilo koji skup usmjerenih grana koji sadrži barem jednu granu iz svake usmjerene putanje od izvora do utoka. Obično postoji mnogo načina da se mreža "prereže". Vrijednost nekog reza se definiše kao zbir kapaciteta grana (u određenom smjeru) reza.

Teorem o maksimalnom protoku i minimalnom rezu tvrdi da, za bilo koju mrežu s jednim izvorom i utokom, *maksimalni izvodivi protok od izvora do utoka jednak je minimalnoj vrijednosti reza* za sve moguće rezove date mreže.

Dakle, ako označimo s F količinu protoka od izvora do utoka za bilo koju šemu protoka, vrijednost bilo kojeg reza pruža gornju granicu za F, a najmanja vrijednost reza jednaka je maksimalnoj vrijednosti F. Stoga, ako se u izvornoj mreži pronađe rez čija vrijednost odgovara vrijednosti F do koje smo do tada došli, trenutna šema protoka mora biti *optimalna*.



Slika 14: Najmanji presjek za naš primjer.

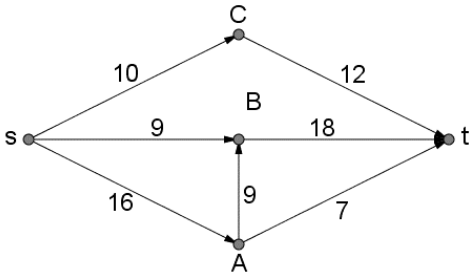
Razmotrimo ponovo mrežu prikazanu na slici 4. Jedan zanimljiv presjek kroz ovu mrežu prikazan je na slici 14. Primijetimo da je vrijednost presjeka $3 + 4 + 1 + 6 = 14$, što je upravo naša maksimalna vrijednost F-a, pa je ovaj presjek minimalni presjek. Također primijetite da u rezidualnoj mreži koja proizlazi iz iteracije 7, gdje je $F = 14$, odgovarajući presjek ima vrijednost nula. Ako bi se to primijetilo, ne bi bilo potrebno tražiti dodatne povećavajuće putanje.

6 Maksimalni protok i minimalni rez: primjeri

6.1 Primjer 1

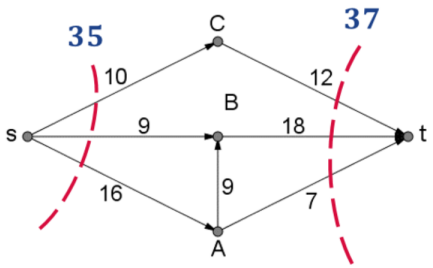
Računanje minimalnog reza

Vrijednost svakog reza se određuje sumiranjem težina svih grana koje su "presječene" a čiji protok ide iz smjera particije grafa koji sadrži čvor izvora ka particiji grafa koji sadrži čvor utok. Minimalni rez je onaj s najmanjom takvom sumom.



Slika 15: Početni graf sa izvorom s i utokom t.

Postoji mnogo rezova koji se mogu izvesti na ovom grafu. Na slici ispod su prikazana dva potencijalna reza. Osim njih, moguće je izvesti, recimo, rez koji siječe grane sC, Bt i At.

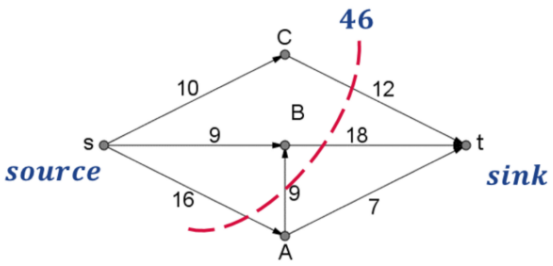


Slika 16: Dva potencijalna reza vrijednosti 35 i 37.

Vrijednost reza dobijamo zbrajanjem težina grana koje su njime presječene, odnosno:

$$35 = 10 + 9 + 16$$
$$37 = 12 + 18 + 7.$$

Osmotrimo i sljedeći rez.



Slika 17: Rez kroz grane Ct, AB i At.

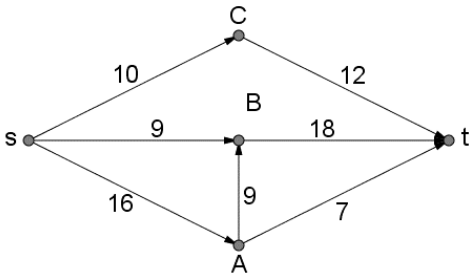
Pri sumiranju težina ne uzimamo u obzir težinu grane AB jer protok kroz istu ide iz dijela grafa koji pripada utoku ka dijelu grafa koji pripada izvoru.

Uz tu opasku, vrijednost ovog reza iznosi $16 + 18 + 12 = 46$.

Nakon što iscrpimo listu svih mogućih rezova zaključujemo da je minimalni mogući rez vrijednosti 35. To znači da i maksimalni protok koji možemo poslat kroz ovu mrežu iznosi 35. Sada ćemo se uvjeriti u to.

Računanje maksimalnog protoka

Posmatrat ćemo jednu po jednu putanju od izvora do utoka i kroz nju ćemo slati maksimalnu količinu protoka u skladu s kapacitetima svih grana.



Slika 18: Isti početni graf - sada pronalazak maksimalnog protoka.

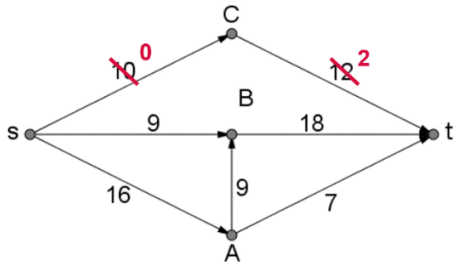
Imamo dvije opcija za praćenje stanja mreže, odnosno dvije alternativne notacije koje možemo koristiti.

Opcija 1: Smanjivati kapacitet grane za količinu poslatog protoka. Nove vrijednosti predstavljaju preostali kapacitet grane, odnosno količine koje se mogu poslati povrh već postojećih.

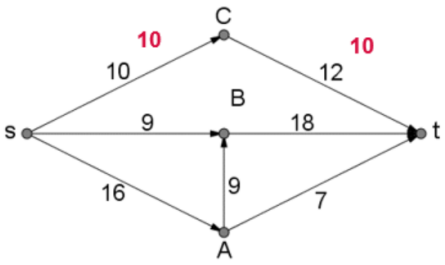
Opcija 2: Pratiti količinu poslatog protoka kroz granu. Razlika ove vrijednosti i kapaciteta grane će biti preostali kapacitet grane.

Ponoviti ovaj postupak za sve putanje koje još nisu dostigle kapacitet nula (što znači da nijedna grana na toj putanji nije u potpunosti iscrpljena). Znači, u svakom koraku biramo jednu putanju koja nema nultih grana, recimo putanju $s - C - t$. Kroz nju možemo poslati protok vrijednosti 10.

Novonastali graf ćemo prikazati kroz obje notacije.



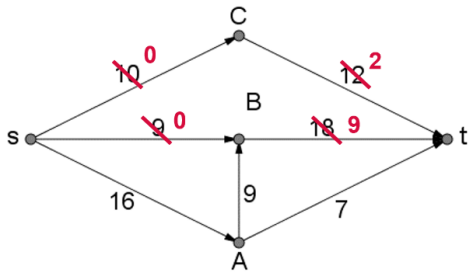
Slika 19: Opcija 1 - smanjujemo kapacitete za količinu poslatog protoka.



Slika 20: Opcija 2 - pored kapaciteta dodajemo informaciju o količini protoka kroz tu granu.

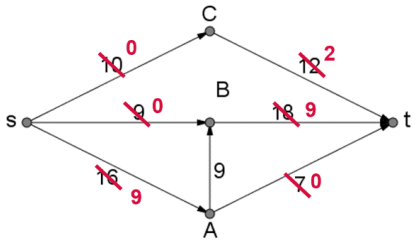
U nastavku ćemo grafove ažurirati koristeći Opciju 1.

Izaberimo sad putanju $s - B - t$. Kroz nju možemo poslati protok 9.



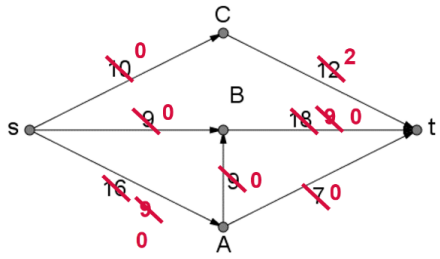
Slika 21: Ažurirani graf nakon slanja protoka 9 kroz putanju $s - B - t$.

Sada biramo putanju $s - A - t$ i šaljemo protok količine 7. Ponovo ažuriramo graf smanjujući kapacitete odgovarajućih grana.



Slika 22: Ažurirani graf nakon slanja protoka 7 kroz putanju $s - A - t$.

Biramo putanju $s - A - B - t$ i šaljemo protok količine 9.



Slika 23: Ažurirani graf nakon slanja protoka 9 kroz putanju $s - A - B - t$.

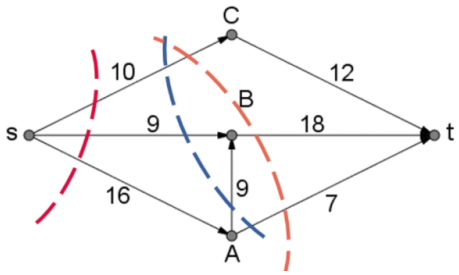
Uočavamo da nam nije preostala niti jedna putanja koja podržava dodatni

protok. Ukoliko sumiramo sve protoke koje smo do sada "pustili", dobivamo:

$$\begin{aligned} s - C - t &= 10 \\ s - B - t &= 9 \\ s - A - t &= 7 \\ s - A - B - t &= 9 \end{aligned}$$

Maksimalni protok iznosi $10+9+7+9 = 35$, što je upravo vrijednost našeg *minimalnog reza* od ranije.

Analizom posljednjeg grafa možemo pronaći odgovarajući minimalni rez. To je zbog toga što, ukoliko je finalni kapacitet grane jednak nuli, kroz nju smo poslali onoliko protoka koliko ona može podnijeti, odnosno "maksimalno smo je iskoristili". To je grana koju bismo metodom *minimalnog reza* u potpunosti presjekli.



Slika 24: Minimalni rezovi koji se mogu izvući iz posljednjeg grafa.

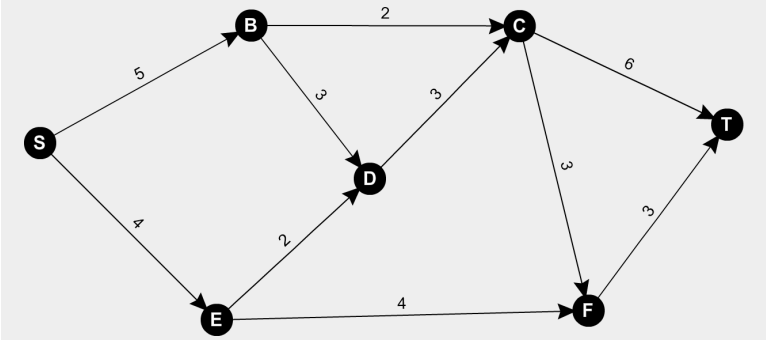
Vrijednosti rezova na Slici 24 su:

$$\begin{aligned} 10 + 9 + 16 &= 35 \\ 10 + 18 + 7 &= 35 \\ 10 + 9 + 9 + 7 &= 35 \end{aligned}$$

6.2 Primjer 2

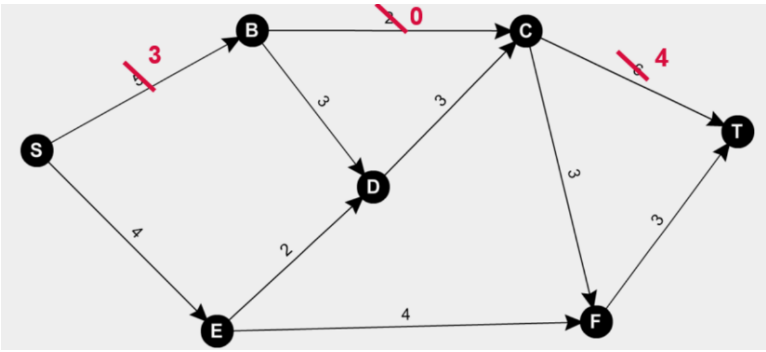
Računanje maksimalnog protoka

Napomenimo da čvor izvor označavamo sa S a čvor utok sa T. Ponovimo algoritam iz primjera 1 na računanje maksimalnog protoka datog grafa.



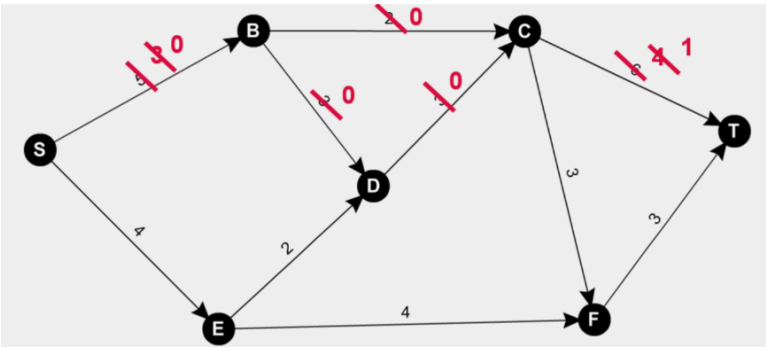
Slika 25: Početni graf za primjer 2.

Biramo putanju kroz koju je moguće poslati neki protok, odnosno putanju koja nema grana s kapacitetom nula. Recimo, kroz putanju $S-B-C-T$ možemo poslati protok količine 2.



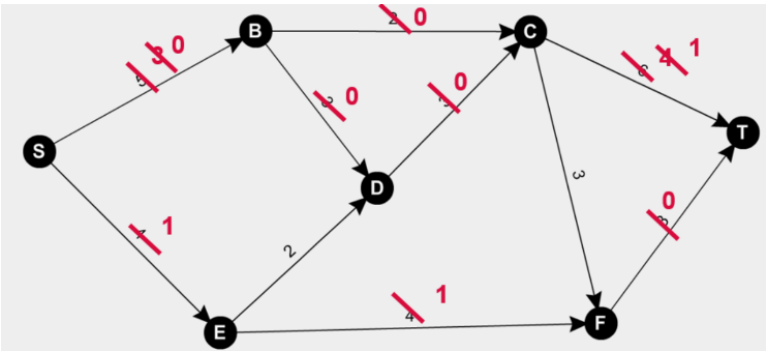
Slika 26: Ažurirani graf nakon slanja protoka 2 kroz putanju $S-B-C-T$.

Biramo putanju $S-B-D-C-T$ čiji je maksimalni protok 3. Ponovo ažuriramo graf smanjujući kapacitete odgovarajućih grana.



Slika 27: Ažurirani graf nakon slanja protoka 3 kroz $S - B - D - C - T$.

Biramo putanju $S - E - F - T$ čiji je maksimalni protok ponovo 3. Shodno tome ažuriramo naš graf.



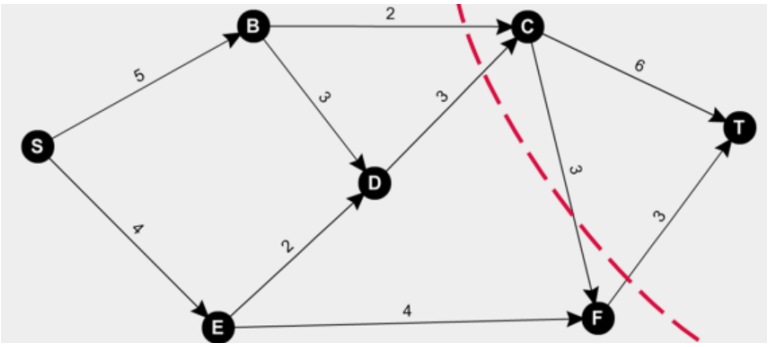
Slika 28: Ažurirani graf nakon slanja protoka 3 kroz putanju $S - E - F - T$.

Dalje ne postoje validne putanje kroz koje možemo dodati protok. Protok cijele mreže dobivamo sumiranjem protoka kroz pojedinačne putanje:

$$\begin{aligned} S - B - C - T &= 2 \\ S - B - D - C - T &= 3 \\ S - E - F - T &= 3 \end{aligned}$$

pa je rješenje $2 + 3 + 3 = 8$.

Kao u prethodnom primjeru, sa zadnjeg grafa možemo identificirati potencijalne minimalne rezove. Oni prolaze kroz grane čije su vrijednosti kapaciteta dosegle nulu.



Slika 29: Minimalni rez siječe grane BC, DC, CF i FT.

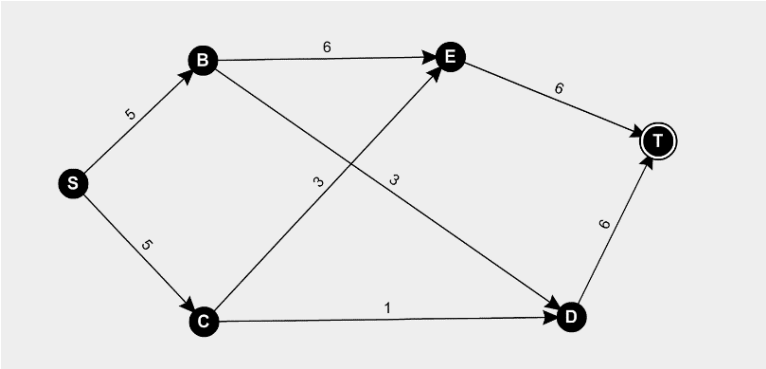
Pri sumiranju težina ne uračunavamo granu CF jer je njen smjer protoka iz područja čvorova koji pripadaju utoku ka čvorovima koji pripadaju izvoru. To zaključujemo iz načina na koji je naš rez "podijelio" graf.

Stoga, vrijednost minimalnog reza je $2+3+3=8$, što je upravo maksimalni mogući protok koji možemo poslati kroz mrežu.

Kako bismo provjerili optimalnost rješenja jednog smjera računanja, naći ćemo rješenje drugog smjera i usporediti. Odnosno, rješavamo dualni problem. Ukoliko se rješenja podudaraju tada smo dosegli optimalnost (slijedi iz prirode primala i duala jednog problema).

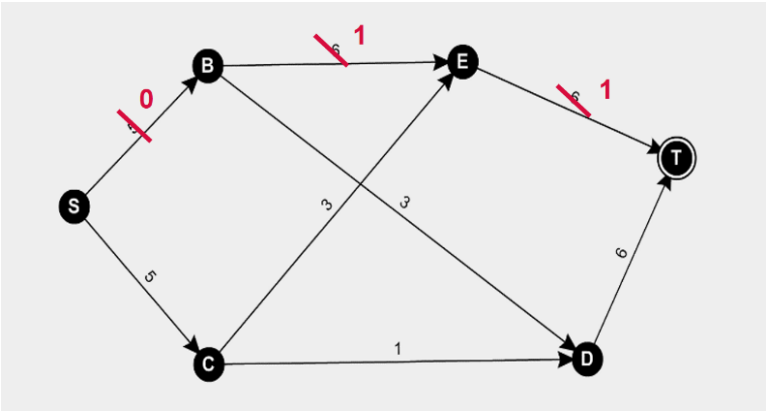
6.3 Primjer 3

Neka nam je dat sljedeći graf sa čvorom izvorom S i čvorom utokom T:



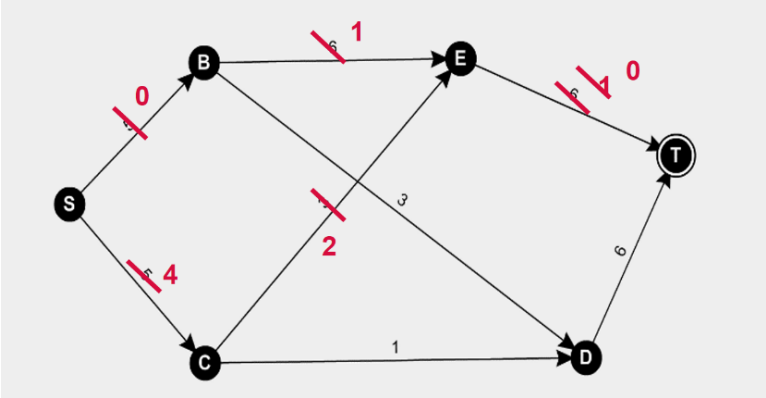
Slika 30: Početni graf na kojem računamo maksimalni protok.

Prateći isti algoritam kao i u ranijim primjerima, biramo jednu putanju s nenultim granama, npr. $S-B-E-T$. Kroz ovu putanju možemo poslati protok količine 5.



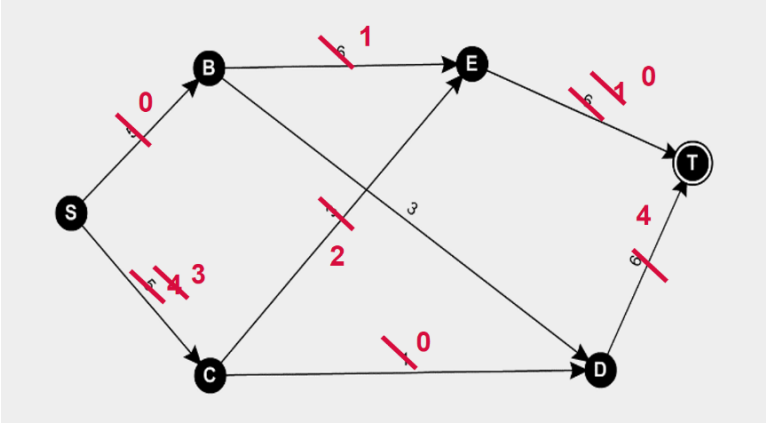
Slika 31: Ažurirani graf nakon slanja protoka 5 kroz putanju $S-B-E-T$.

Sada biramo putanju $S-C-E-T$ kroz koju možemo pustiti protok količine 1.



Slika 32: Ažurirani graf nakon slanja protoka 1 kroz putanju $S-C-E-T$.

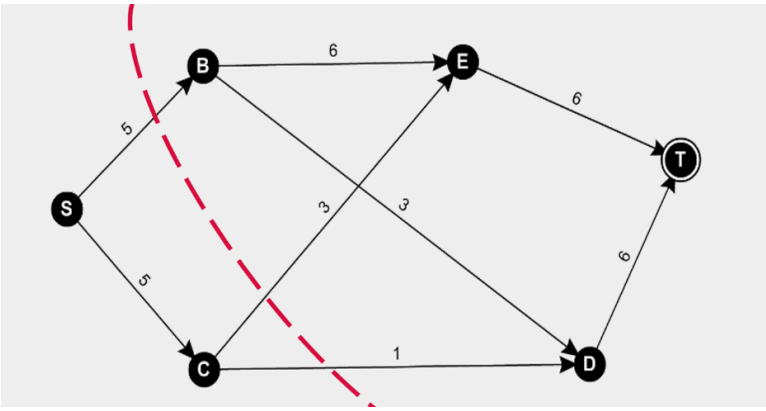
Biramo putanju $S-C-D-T$ i ponovo šaljemo protok količine 1.



Slika 33: Ažurirani graf nakon slanja protoka 1 kroz putanju $S-C-D-T$.

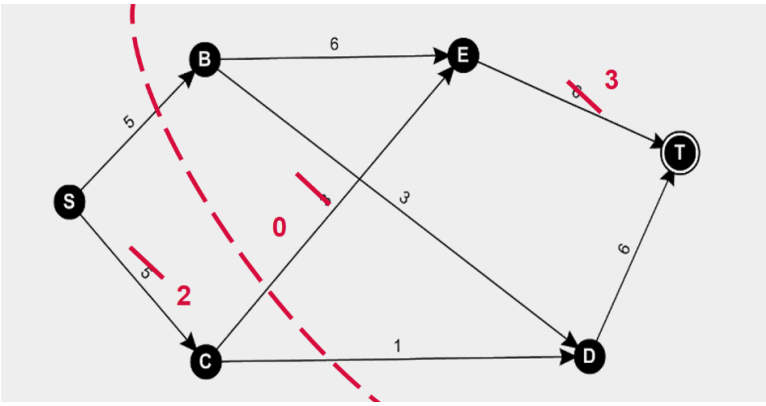
Iscrpili smo sve putanje sa nenultim granama i dosegli ukupnu vrijednost protoka od izvora do utoka: $5+1+1=7$.

Uočimo da je u posljednjem grafu nemoguće identificirati minimalni rez s obzirom da niti jedan rez neće proći kroz sve nulte grane. To nam je znak da nemamo optimalno rješenje. Pronađimo minimalni rez iscrpnom pretragom vrativši se na polazni graf.



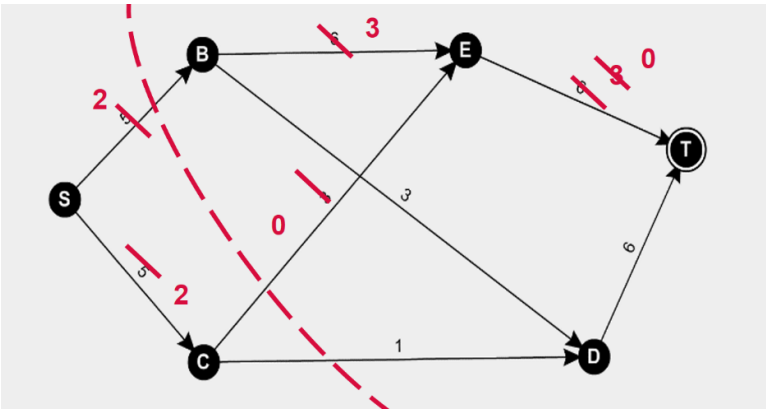
Slika 34: Zaključeni minimalni rez tehnikom iscrpne pretrage nad polaznim grafom.

Vrijednost minimalnog reza je $5 + 3 + 1 = 9$ i vidimo da isti siječe granu CE koja je u prethodnom slučaju imala neiskorišteni kapacitet 2 (jer smo od ukupnog kapaciteta 3 iskoristili samo 1). Vratimo se na polazni graf i pošaljimo sve 3 jedinice kroz putanju $S-C-E-T$. Dobivamo iduće stanje:



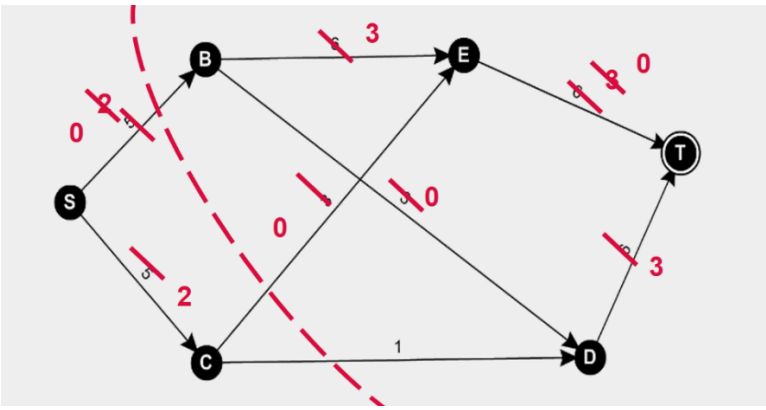
Slika 35: Pronađeni minimalni rez i preostali kapaciteti grana nakon slanja protoka 3 kroz putanju $S - C - E - T$.

Regularno nastavljamo naš algoritam birajući putanju s nenultim granama, recimo $S - B - D - T$, i šaljemo količinu protoka 3.



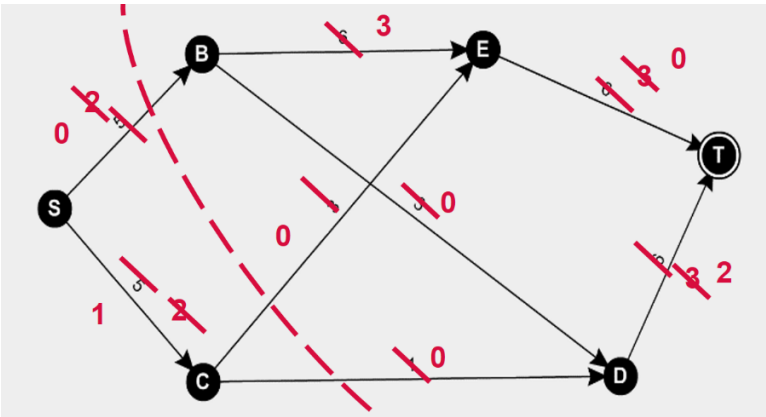
Slika 36: Ažurirani graf nakon slanja protoka 3 kroz putanju $S - B - E - T$.

Biramo putanju $S - B - D - T$ i šaljemo protok 2.



Slika 37: Ažurirani graf nakon slanja protoka 2 kroz putanju $S - B - D - T$.

Biramo putanju $S - C - D - T$ i šaljemo protok 1.



Slika 38: Ažurirani graf nakon slanja protoka 1 kroz putanju $S-C-D-T$.

Ukupni protok koji smo poslali kroz mrežu iznosi:

$$S - C - E - T = 3$$
$$S - B - E - T = 3$$
$$S - B - D - T = 2$$
$$S - C - D - T = 1$$

$3 + 3 + 2 + 1 = 9$, što je također vrijednost našeg *minimalnog reza*, stoga smo dosegli optimalno rješenje.

7 Problem maksimalnog protoka: Algoritmi

Problem maksimalnog protoka je, zbog svoje aktuelnosti, predmet intenzivnog proučavanja. Stalno se razvijaju sve bolji algoritmi za njegovo rješavanje.

Konceptualno najjednostavniji specijalistički algoritam za rješavanje problema maksimalnog protoka je Ford-Fulkersonov algoritam.

7.1 Ford-Fulkersonov algoritam

Ideja ovog algoritma je da se, polazeći od nekog dopustivog protoka, kroz iteracije vrijednost ukupnog protoka stalno povećava duž određenih putanja u grafu od izvora do utoka sve dok se ne postigne maksimalna moguća vrijednost ukupnog protoka. Početni dopustivi protok se može naći ili nekom empirijskom metodom, ili se prosto može proći od trivijalnog protoka kod kojeg su protoci kroz svaku od grana jednaki nuli (ovakav protok je očigledno dopustiv).

Ford i Fulkerson nisu tačno precizirali kojim se tačno redoslijedom vrši povećavanje protoka, nego su to smatrali "implementacionim detaljem". Stoga postoji veliki broj različitih implementacija (verzija) ovog algoritma. Jedna od tih verzija je Edmonds-Karpova.

Ford-Fulkersonov algoritam

Input: Graf $G = (V, E)$ s kapacitetom protoka c, čvorom izvorom s, i čvorom utokom t.
Output: Maksimalni protok f iz s u t.

1: $f(u, v) \leftarrow 0$ za sve grane (u, v)
2: **while** postoji put p od s do t u G_f, takav da $c_f(u, v) > 0$ za svaku granu $(u, v) \in p$ **do**
3: Naći $c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$
4: **for** svaku granu $(u, v) \in p$ **do**
5: $f(u, v) \leftarrow f(u, v) + c_f(p)$ {Poslati protok kroz ovu granu.}
6: $f(v, u) \leftarrow f(v, u) - c_f(p)$ {Protok se možda "vrati" kasnije.}
7: **end for**
8: **end while**

Put u koraku 2 može se pronaći primjerice pretraživanjem u širinu (BFS) ili pretraživanjem u dubinu (DFS) u $G_f(V, E_f)$. Prvu varijantu koristi Edmonds–Karpov algoritam.

Složenost

Dodajući pojačavajuće putanje protoku koji je već utvrđen u grafu doći ćemo do maksimalnog protoka kada više ne budemo mogli da pronađemo nove pojačavajuće putanje. Međutim, ne postoji sigurnost da će se do ove situacije ikad doći, tako da je jedino sigurno da ukoliko se algoritam zaustavi da će odgovor biti tačan. U slučaju da algoritam radi beskonačno, može se desiti da protok uopšte ne konvergira ka maksimalnom protoku. Ali, ova situacija se događa samo sa iracionalnim vrijednostima protoka. Kada su kapaciteti cijeli brojevi, vrijeme izvršavanja je ograničeno sa $O(Ef)$ gde je E broj grana u grafu i f maksimalan protok grafa. Ovo je zato što se pojačavajuća putanja može pronaći u $O(E)$ vremena i povećati protok za cjelobrojnu vrijednost koja je minimum 1.

Varijacija Ford-Fulkerson algoritma koja garantuje završetak i vremenski je nezavisna od maksimalnog protoka u grafu je Edmonds-Karp algoritam koji radu vremenu $O(VE^2)$

7.2 Edmonds-Karpova verzija

Varijanta Ford-Fulkersonovog algoritma koja se danas najviše koristi u praksi poznata je pod nazivom Edmonds-Karpova verzija Ford-Fulkersonovog algoritma ili ponekad samo Edmonds-Karpov algoritam, mada je jasno da se radi samo o specijalnoj verziji Ford-Fulkersonovog algoritma. U ovoj varijanti, u svakoj iteraciji se povećanje protoka vrši isključivo putem najkraćeg od svih mogućih povećavajućih putanja od izvora do utoka (ili nekim od njih ukoliko takvih putanja ima više), pri čemu se pod dužinom putanje prosto smatra broj grana u putanji (a ne zbir njihovih težina). Najkraća povećavajuća putanja se traži BFS pretragom.

Implementacija u Pythonu

```
1  import collections
2
3  class Graph:
4    """
5    Ova klasa predstavlja reprezentaciju usmjerenog grafa
     preko matrice susjedstva.
6    """
7
8    def __init__(self, graph):
9      self.graph = graph #rezidualni graf
```

```
10        self.row = len(graph)
11
12      def bfs(self, s, t, parent):
13
14        """
15        Vraca true ako postoji put od izvora 's' do utoka '
      t' u rezidualnom grafu. Takodjer popunjava parent[]
      kako bi sacuvao putanju.
16        """
17
18        # Oznacava sve cvorove koji nisu posjeceni
19        visited = [False] * self.row
20
21        # Kreira red za BFS
22        queue = collections.deque()
23
24        # Oznacava izvor kao posjecen i izbacuje ga iz reda
25        queue.append(s)
26        visited[s] = True
27
28        # Standardna BFS petlja
29        while queue:
30          u = queue.popleft()
31
32          # Uzima sve susjede cvora kojeg smo izbacili iz
      reda (cvor u). Ako susjed nije posjecen onda ga
      oznacavamo kao posjecenog i izbacujemo iz reda
33          for ind, val in enumerate(self.graph[u]):
34            if (visited[ind] == False) and (val > 0):
35              queue.append(ind)
36              visited[ind] = True
37              parent[ind] = u
38
39        # Ako smo dosli to utoka u BFS-u pocevsi of izvora,
       onda vracamo true, inace false
40        return visited[t]
41
42      def edmonds_karp(self, source, sink):
43        # Ovaj niz popunjava BFS, cuvaju se putanje
44        parent = [-1] * self.row
45
46        max_flow = 0  # Nema protoka u pocetku
47
```

```
48      # Sve dok postoji put od izvora do utoka povecavaj protok
49      while self.bfs(source, sink, parent):
50        # Nadji minimalni preostali kapacitet grana na putanji koju je popunio BFS. Odnosno, nadji maksimalni protok kroz tu putanju.
51        path_flow = float("Inf")
52        s = sink
53        while s != source:
54          path_flow = min(path_flow,
55                self.graph[parent[s]][s])
56          s = parent[s]
57
58        # Dodaj trenutni protok na ukupni protok
59        max_flow += path_flow
60
61        # Azuriraj preostale kapacitete grana i obrnutih grana na putanji
62        v = sink
63        while v != source:
64          u = parent[v]
65          self.graph[u][v] -= path_flow
66          self.graph[v][u] += path_flow
67          v = parent[v]
68
69      return max_flow
70
71  # PRIMJER 1
72  matrica1 = [
73    [0, 16, 13, 0, 0, 0],
74    [0, 0, 10, 12, 0, 0],
75    [0, 4, 0, 0, 14, 0],
76    [0, 0, 9, 0, 0, 20],
77    [0, 0, 0, 7, 0, 4],
78    [0, 0, 0, 0, 0, 0]
79  ]
80
81  graf1 = Graph(matrica1)
82  izvor = 0
83  utok = 5
84  maks_protok1 = graf1.edmonds_karp(izvor, utok)
85  print("Maksimalni protok 1:", maks_protok1)
86
```

```
87  # PRIMJER 2
88  matrica2 = [
89    [0, 10, 0, 8, 0, 0],
90    [0, 0, 5, 2, 0, 0],
91    [0, 0, 0, 0, 0, 7],
92    [0, 0, 0, 0, 10, 0],
93    [0, 0, 8, 0, 0, 10],
94    [0, 0, 0, 0, 0, 0]
95  ]
96
97  graf2 = Graph(matrica2)
98  izvor = 0
99  utok = 5
100 maks_protok2 = graf2.edmonds_karp(izvor, utok)
101 print("Maksimalni protok 2:", maks_protok2)
102
```

Danas postoje i znatno efikasniji ali i znatno složeniji algoritmi za nalaženje maksimalnog protoka. Neki od njih su **Dinitzov algoritam i Push-Relabel algoritam**.

Svaki algoritam ima svoje karakteristike i prednosti u različitim scenarijima i vrstama grafova. Na primjer, Dinitzov algoritam također koristi BFS, ali na drugačiji način od Edmonds-Karp algoritma, dok Push-Relabel algoritam koristi složenije tehnike za praćenje protoka kroz graf. Odabir algoritma ovisi o specifičnim zahtjevima problema i performansama koje se očekuju.

7.3 Dinitzov algoritam

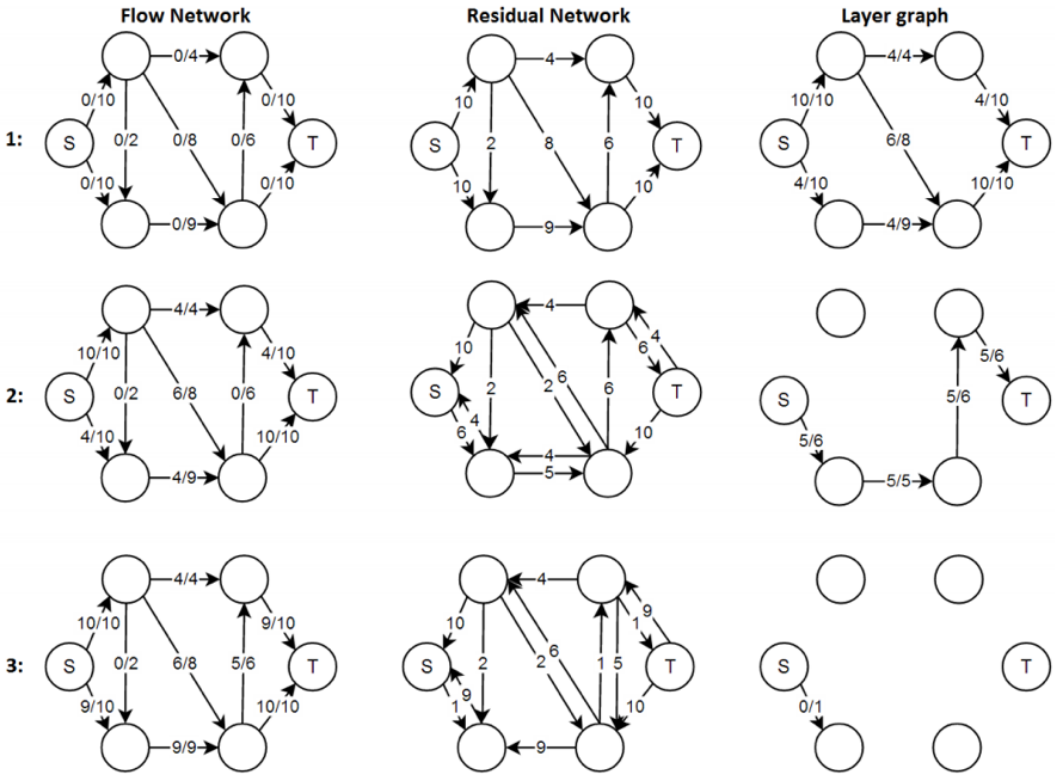
U Edmonds-Karp algoritmu, koristimo BFS kako bismo pronašli povećavajući put i poslali protok duž tog puta. U Dinitzovom algoritmu, pomoću istog provjeravamo je li moguć dodatni protok i konstruišemo **level-graf**. U level-grafu dodjeljujemo levele svim čvorovima, te ga koristimo da kroz grane šaljemo protoke. **Level čvora** je najkraća udaljenost (u broju grana) tog čvora od izvora. U Edmonds-Karpu, šaljemo samo protok duž puta pronađenog BFS-om.

Dinitzov algoritam

1: Inicijalizirati rezidualni graf G.
2: Iskoristiti BFS od G kako bismo konstruirali level-graf (ili dodijelili levele čvorovima) i također provjeriti je li moguć dodatni protok.
3: **if** dodatni protok nije moguć **then**
4: **return**.
5: **end if**
6: Šaljemo višestruke protoke kroz G koristeći level-graf dok se ne dostigne blokirajući protok.

Pod "koristeći level-graf" podrazumijevamo da u svakom protoku leveli čvorova trebaju biti $0, 1, 2, ...$ (redom) od čvora s do čvora t.

Protok je **blokirajući** ako je nemoguće poslati dodatke protoke koristeći level-graf, tj. ne postoji više $s - t$ putanja takvih da čvorovi takvih putanja imaju trenutne levele $0, 1, 2, ...$ redom.



Slika 39: Dinitzov algoritam

7.4 Push-Relabel algoritam

Ford-Fulkerson: sličnosti

Kao i Ford-Fulkerson, Push-Relabel algoritam također djeluje nad rezidualnom grafu (Rezidualni graf je graf koji predstavlja sve moguće dodatne protoke. Ako postoji put od izvora do utoka u rezidualnom grafu, tada je moguće dodati protok).

Ford-Fulkerson: razlike

Push-Relabel algoritam radi lokalizirano. Umjesto da proučava cijeli rezidualni graf za pronalazak povećavajuće putanje, ovaj algoritam djeluje jedan po jedan čvor. U Ford-Fulkersonu, neto razlika između ukupnog odliva robe i ukupnog priliva se održava na 0 za svaki čvor sem izvora i utoka. Push-Relabel algoritam dozvoljava da priliv premaši odliv robe prije nego dođemo do konačnog rješenja protoka kroz mrežu. U konačnom protoku je neto razlika ponovo 0 za svaki čvor sem za čvor izvor s i čvor utok t.

Algoritam

Svakom čvoru je pridružena visina i višak protoka. Visina čvora nam govori da li taj čvor može poslati protok ka idućem čvoru ili ne. Protok je moguć samo iz čvora veće visine u čvor manje visine (iz višeg čvora u niži). Višak protoka je razlika ukupnog priliva roba u čvor i ukupnog odliva koji izlazi iz tog čvora.

Push-Relabel algoritam

```
1: Pred-protok(): Inicijalizira protoke i visine
2: while moguće je sprovesti Push() ili Relabel() nad čvorom do
3:    // ili dok postoji čvor sa viškom protoka
4:    uradi Push() ili Relabel()
5: end while
6: // u ovom trenutku svi čvorovi imaju višak protoka 0 (osim
   izvora i utoka)
7: return protok
```

Postoje tri glavne operacije u Push-Relabel algoritmu:

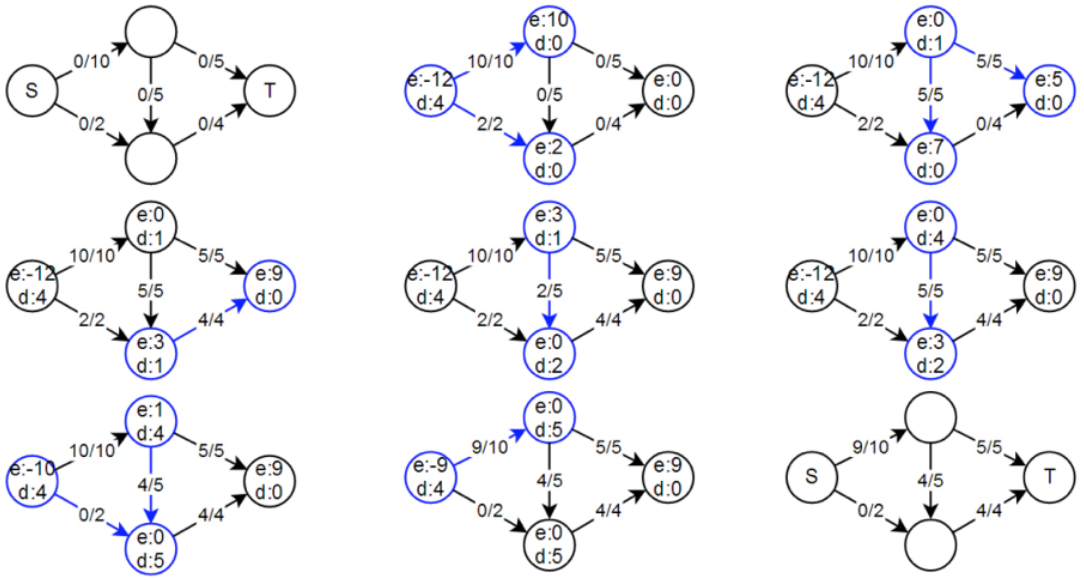
1. **Pred-protok():** Inicijalizira visine i protoke svih čvorova.

Pred-protok()

1: Inicijaliziraj visinu i protok svakog čvora na 0.
2: Inicijaliziraj visinu izvora na ukupni broj čvorova u grafu.
3: Inicijaliziraj protok svake grane na 0.
4: Za sve susjedne čvorove od izvora s vrijednosti protoka i viška protoka inicijalizirati na kapacitet.

2. **Push():** Koristi se za slanje protoka iz čvora koji ima višak protoka. Ako čvor ima višak protoka i susjeda manje visine (u rezidualnom grafu), šaljemo protok ka tom susjedu. Količina koja se šalje je jednaka minimumu viška protok i kapaciteta grane.

3. **Relabel():** Koristi se kada čvor ima višak protoka i nema susjeda manje visine. Povećavamo visinu izabranog čvora tako da možemo izvesti operaciju Push(). Da bismo to uradili, biramo susjedni čvor s minimalnom visinom (u rezidualnom grafu) i našem čvoru pridružujemo njegovu visinu + 1.



Slika 40: Push-Relabel algoritam

8 Python biblioteka NetworkX

NetworkX je Python biblioteka koja se koristi za stvaranje, manipulaciju i proučavanje složenih mreža ili grafova. Pruža opsežan skup alata za rad s grafovima, uključujući algoritme za rješavanje različitih problema vezanih uz grafove, poput problema maksimalnog protoka.

Pruža nekoliko algoritama za rješavanje problem maksimalnog protoka, uključujući Ford-Fulkersonov, Edmonds-Karpov i Dinitzov algoritam. Uglavnom koristi Edmonds-Karpov algoritam za rješavanje problema maksimalnog protoka, te je on implementiran u funkciji `maximum_flow`.

8.1 Primjer rješavanja problema

Kod u nastavku koristi spomenutu biblioteku i biblioteku **Matplotlib** kako bi izračunao maksimalni protok u usmjerenom grafu i vizualizirao rezultate.

Prvo se uvoze navedene biblioteke a zatim se definira matrica susjedstva koja predstavlja kapacitete grana kroz koji šaljemo protok.

Grane kapaciteta različitog od nula se dodaju u graf koristeći petlje koje prolaze kroz matricu susjedstva.

Korištenjem funkcije `nx.maximum_flow(G, 0, 5)` izračunava se maksimalni protok u grafu, pri čemu je 0 čvor izvor a 5 čvor utok.

Ispisuje se maksimalni protok i protok kroz svaku granu.

Vizualizacija grafa se izvodi pomoću Matplotlib-a. Postavlja se veličina figure, izračunavaju se pozicije čvorova koristeći `nx.spring_layout`, zatim se crta graf s oznakama čvorova i grana. Oznake sadrže protok preko grane i kapacitete grana. Na kraju se prikazuje graf pozivom `plt.show()`.

```
import networkx as nx
import matplotlib.pyplot as plt

# Definisemo matricu susjedstva (redovi su cvorovi,
 kolone su kapaciteti grana)

matrica = [
  [0, 16, 13, 0, 0, 0],
  [0, 0, 0, 12, 0, 0],
  [0, 4, 0, 0, 14, 0],
  [0, 0, 9, 0, 0, 20],
  [0, 0, 0, 7, 0, 4],
  [0, 0, 0, 0, 0, 0]
]

# Kreiramo usmjereni graf
G = nx.DiGraph()

# U graf dodajemo grane iz matrice susjedstva
for i in range(len(matrica)):
  for j in range(len(matrica[i])):
    if matrica[i][j] != 0:
      G.add_edge(i, j, capacity=matrica[i][j])

# Racunamo maksimalni protok
maks_protok, mapa_protoka = nx.maximum_flow(G, 0, 5)

print("\nMaksimalni protok: ", maks_protok)

print("\nProtok kroz grane:")
for u in mapa_protoka:
  for v in mapa_protoka[u]:
    print(f"Grana ({u}, {v}): {mapa_protoka[u][v]}")

print()

# Vizualizacija
plt.figure(figsize=(10, 6))
pos = nx.spring_layout(G)

nx.draw(G, pos, with_labels=True,
node_color='lightblue', node_size=1500, font_size=12)

```

```
43  oznake_grana = { (u, v):
44    f"{mapa_protoka[u][v]}/{G[u][v]['capacity']}"
45    for u, v in G.edges() }
46
47  nx.draw_networkx_edge_labels(G, pos,
48              edge_labels=oznake_grana)
49
50  plt.title("Maksimalni protok sa NetworkX i Matplotlib")
51  plt.show()
52
```

Iako implementacija NetworkX-a uglavnom koristi Edmonds-Karp, može također uključivati poboljšanja i optimizacije iz drugih algoritama, ali je osnovni pristup temeljen na Edmonds-Karpu.

9 Sažetak

Mnoge probleme u operacionim istraživanjima možemo modelirati kao mrežne probleme, koji su primjenjivi u raznim kontekstima. Fokus je na problemima protoka, uz detaljniju obradu problema minimalnog troška i njegovim specijalnim slučajem - problemom maksimalnog protoka.

Predstavljene su totalno-unimodularne matrice te njihova veza s cjelobrojnim rješenjima linearnih programa. Pokazano je da je matrica čvor-grana u problemu protoka totalno-unimodularna, što osigurava da će optimalna rješenja biti cjelobrojna. Ovo svojstvo omogućava da se LP relaksacija problema protoka koristi za dobivanje cjelobrojnih rješenja, čak i kada granice varijabli nisu eksplicitno cjelobrojne.

Specijalni slučajevi problema protoka minimalnog troška obuhvaćaju nekoliko varijacija ovog osnovnog problema. Problem najkraćeg puta formulira se kao traženje najkraćeg puta od početnog čvora do krajnjeg čvora u neusmjerenom grafu. Problem transporta odnosi se na prevoz robe od čvorova izvora do čvorova utoka, distribuirajući određenu količinu robe prema potrebama, bez postavljanja gornjih granica na pojedinačne tokove. Ovaj problem je čest u logistici, gdje je cilj efikasno rasporediti robu između različitih lokacija uz minimalne troškove prijevoza. Problem asignacije se može posmatrati kao posebna vrsta problema transporta, uz uslove da je broj čvorova izvora jednak broju čvorova utoka i da se svakom čvoru izvora pridružuje količina 1, dok se svakom čvoru utoka pridružuje količina -1. Ovaj problem se često koristi u ekonomiji i operacionim istraživanjima za raspoređivanje resursa.

Problem maksimalnog protoka traži maksimalni protok od izvora do utoka u usmjerenom grafu, uz zadovoljavanje određenih ograničenja: protok kroz svaku granu ne smije prelaziti dati kapacitet, a količina koja ulazi u čvor mora biti jednaka količini koja izlazi iz istog čvora, osim za izvor i utok. Ovaj problem ima širok spektar primjena u telekomunikacijskim mrežama, transportnim sistemima, logistici, proizvodnji, energetskim mrežama i biološkim sistemima.

Algoritam za rješavanje problema maksimalnog protoka temelji se na pronalaženju povećavajućih putanja u rezidualnom grafu i dodjeljivanju protoka duž tih putanja. Nakon svake iteracije, provjerava se postojanje povećavajućih putanja sve dok se ne postigne maksimalni protok. Pronalaženje povećavajućih putanja može biti izazovno u velikim mrežama, ali se može pojednostaviti

sistematičnim postupkom. Primjena teorema o maksimalnom protoku i minimalnom rezu omogućava provjeru optimalnosti pronađene šeme protoka.

Kroz tri primjera detaljno je prikazan postupak za pronalaženje maksimalnog protoka u mrežama. Tekst se potom okreće analizi algoritama. Opisani su Ford-Fulkersonov algoritam, Edmonds-Karpov (čija je data i implementacija), Dinitzov algoritam te Push-Relabel algoritam. Date su njihove opće karakteristike, pseudokodovi i ilustracije.

Za kraj, predstavljena je Python biblioteka NetworkX za rješavanje problema na mreži. Uz vizualizacije biblioteke Matplotlib, rješen je konkretan primjer maksimalnog protoka na datoj matrici susjedstva.

Literatura

[1] Wolsey, L.A. *Integer Programming.* New York: Wiley, 1998.

[2] Hillier, F.S., Lieberman, G.J. *Introduction to Operations Research.* McGraw-Hill, 2001.

[3] Fourer, R. *Optimization Methods.* Northwestern University, 2005.

[4] Mateljan, T., Jurić, Ž. *Osnove operacionih istraživanja.* Univerzitet u Sarajevu, 2015.

[5] Bazaraa, M.S., Jarvis, J.J., Sherali, H.D. *Linear Programming and Network Flows.* Hoboken, N.J.: John Wiley & Sons, 2010.

[6] Vanderbei, R.J. *Linear Programming: Foundations and Extensions.* New York, N.Y: Springer, 2001.

[7] Cormen, T., Leiserson, C., Rivest, R., Stein, C. *Introduction to Algorithms.* Third ed. Cambridge, Mass.: Mit Press, 2009.

[8] Goldberg, A.V., Tarjan, R.E. *A new approach to the maximum flow problem.* Proceedings of the eighteenth annual ACM symposium on Theory of computing - STOC '86, 1986.

[9] Even, S., Goldreich, O., Rosenberg, A.L., Selman, A.L. *Theoretical computer science : essays in memory of Shimon Even.*. Berlin: Springer., 2006

[10] Kleinberg, J., Éva Tardos *Algorithm design.* Harlow, Essex: Pearson., 2014

[11] Heineman, G.T., Pollice, G., Selkow, S. *Algorithms in a nutshell.* Sebastopol, Ca: O'reilly, Cop., 2016

[12] Anderson, J.A., Lewis, J., O Dale Saylor *Discrete mathematics with combinatorics.* Upper Saddle River, N.J.: Pearson Education., 2004

[13] Bernhard Korte, Jens Vygen, Springerlink *Combinatorial Optimization: Theory and Algorithms.* Berlin, Heidelberg: Springer Berlin Heidelberg., 2012