SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI RAD br. 430

# Problem pridruživanja: rješenja i primjene

Kristijan Verović

Zagreb, lipanj 2022.

*Zahvaljujem mentorima prof. dr. sc. Marinu Šiliću i dr. sc. Adrianu Satji Kurdiji na strpljenju i korisnim savjetima prilikom izrade rada.*

*Zahvaljujem se obitelji i prijateljima na podršci i lektoriranju.*

# SADRŽAJ

# 1. Uvod

Problem pridruživanja(engl. *Assigment problem*) odnosno njegov posebni slučaj sparivanje je čest problem s kojim se ljudi susreću, bilo to sparivanje staklenki i poklopaca za njih, maksimalno sparivanje prijavnika za posao i otvorenih pozicija, najveći broj nogometnih momčadi koje mogu igrati u subotu pod uvjetom da one sparene nisu igrale zadnje 3 utakmice zajedno i slično. Manje eksplicitno, postoje problemi koji se mogu svesti na problem pridruživanja za što ima primjera [3] i [4].

Mnoge posebne slučajeve problema pridruživanja, pa i problema sparivanja koji je i sam poseban slučaj problema pridruživanja možemo riješiti ad-hoc pristupima različite male polinomne složenosti ovisno o problemu, međutim ad-hoc pristup nije uvijek moguć te nam treba općeniti algoritam. Najnaivniji pristup je isprobavati sve kombinacije sparivanja što je faktorijalne složenosti te je kao takvo neupotrebljivo za imalo veće skupove.

U ovom tekstu ćemo precizno definirati problem pridruživanja, varijante, izvesti algoritam te navesti neke primjene. Koristit ćemo maxflow algoritam(Edmonds–Karp varijantu) na prilagođenom bipartitnom grafu kako bi našli maksimalno bestežinsko sparivanje u $O(n * m^2)$ vremenskoj složenosti. Nakon osnovne verzije problema razmatramo težinsko spajanje te nalazimo algoritam koji rješava taj problem popraćen dokazom. Izvedeni algoritam poznat je kao mađarski algoritam(engl. *Hungarian algorithm*).

Mađarski algoritam je $O(n^3)$ vremenske složenosti te $O(n^2)$ prostorne složenosti odnosno moguća je i $O(n)$ prostorna složenost ukoliko se težine između elemenata skupova mogu izračunati temeljem jednostavne kalkulacije svaki put(na primjer točke u 2d prostoru i euklidska udaljenost među njima).

Povijesno gledano interes za protoke kroz mrežu te varijante istih počeo je oko 1950-ih u SAD-u. Tako 1955. nastaje djelo [7] koje rješava problem pridruživanja u $O(n^4)$. Nazvano je mađarskom metodom jer se ideja temeljila na dokazu "König-Egeváry" teorema dva mađarska matematičara iz 1931. Kasnije je došlo (barem službeno) unaprjeđenje algoritma manjim modificiranjem u $O(n^3)$. Otprilike istovremeno

pedesetih godina prošlog stoljeća nastaje Ford Fulkersonov algoritam za maksimalni protok kroz graf. Svi ovi algoritmi se na ovaj ili onaj način oslanjaju na ideju "poboljšavanje putova"(engl. *augmenting paths*), u svrhu pojačavanja protoka kroz mrežu. Otkriveno je bilo poslije da je u 19. stoljeću njemački matematičar Carl Gustav Jacobi pronašao polinomno rješenje problema pridruživanja u svrhu rješavanja diferencijalnih jednadžbi [1].

# 2. Problem pridruživanja

## 2.1. Definicija problema

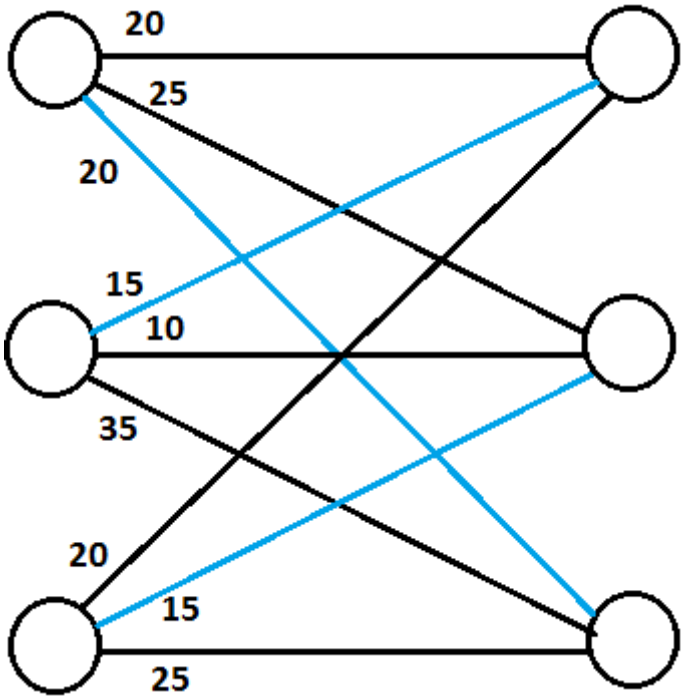
Neka je M matrica cijelih brojeva dimenzije NxN, nek je S skup svih permutacija brojeva od 1 do N. Neka je $g : S \mapsto \mathbb{Z}$, funkcija definirana kao $g(V) = \sum_{i=1}^{n} M[i][V[i]], V \in S$. Rješenje problema pridruživanja je onaj $K \in S$, za koji vrijedi $g(K) \leq g(V), \forall V \in S$. Ovdje treba primijetiti da rješenje potencijalno nije jedinstveno.

Manje formalno, idemo inkrementalno po recima matrice uzimajući neku vrijednost iz stupca iz kojeg do sad nismo uzimali. Takvih uzimanja ima N! što je predstavljeno skupom $S$. Od svih tih uzimanja želimo uzeti onu koja daje najmanji mogući zbroj elemenata matrice. Na tablici 2.1 se može vidjeti jedan riješeni primjer. Redoslijed permutacija 3->1->2 daje minimalno težinsko sparivanje vrijednosti 50 te je označena u plavom. Bitno je primijetiti da kombinacija 3->2->1 isto daje optimalno rješenje vrijednosti 50. Ostale permutacije daju veću vrijednost od 50 i stoga nisu optimalne, na primjer 1->3->2 daje 70.

**Tablica 2.1:** Matrica sa sparivanjem minimalne težine.

| | | |
|---|---|---|
| 20 | 25 | 20 |
| 15 | 10 | 35 |
| 20 | 15 | 25 |

Ekvivalentan prikaz matričnom je prikaz preko težinskog bipartitnog grafa, čiji prikaz je i prirodna motivacija problema pridruživanja. Plavom bojom su označeni odabrani bridovi.




**Slika 2.1:** Težinski bipartitni graf ekvivalentan prikazu u tablici 2.1.

Rješenje problema pridruživanja je potpuno sparivanje i to ono čija je suma bridova najmanja. Dobivanje samog potpunog sparivanja u potpunom bipartitnom grafu je trivijalno, međutim dobivanje baš "najlakšeg" sparivanja nije jednostavno.

## 2.2. Varijante i primjene

Nakon definicije slijede posebni slučajevi i varijante problema pridruživanja kao i primjene istih. To bi trebalo više približiti motivacije cijele problematike čitatelju.

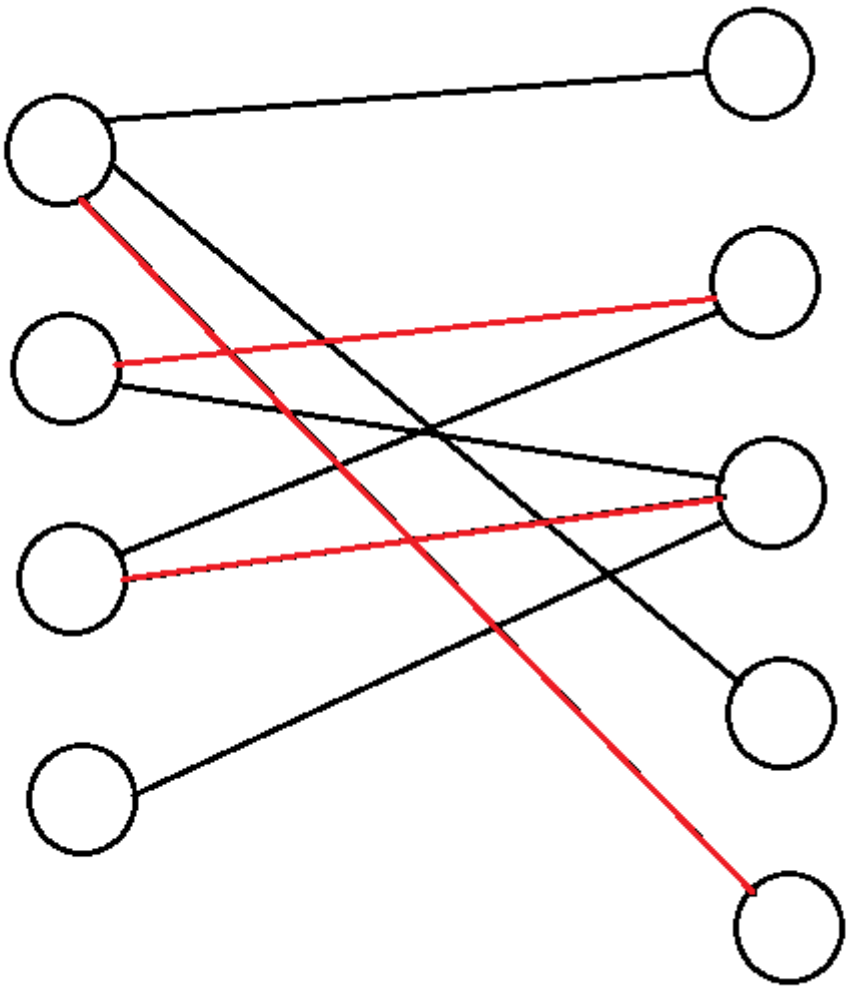
### 2.2.1. Bestežinsko sparivanje

Sparivanje u kojem bridovi nemaju težine nego je samo bitno da postoje.

#### Sparivanje maksimalnog kardinaliteta

Najraširenija varijanta koja zapravo prethodi problemu pridruživanja je problem maksimalnog sparivanja u bipartitnom grafu. U njemu želimo spariti međusobno maksimalan broj vrhova iz lijeve i desne skupine u bipartitnom grafu tako da niti jedan spareni vrh nije sparen s nekim drugim. Problem je u tome da bipartitni graf nije potpuno sparen te se moraju pametno birati sparivanja ne bi li se ostvarilo ono maksimalno. Slijedi

primjer na slici 2.2, crvenom bojom su označena iskorištena sparivanja.



**Slika 2.2:** Bipartitni graf i njegovo najveće sparivanje.

Ovaj problem se svodi na rješavanje problema pridruživanja: pretvorimo postojeće veze u 1 u matrici, a nepostojeće u 0. Tablica 2.2 to prikazuje, spareni bridovi su označenim crvenom bojom u tablici. Zadnji redak je ispunjen nulama iako petog lijevog čvora nema kako bismo ostali u definiciji problema pridruživanja. Slabijom crvenom su ostvarena fiktivna sparivanja unutar definicije problema pridruživanja, suma sparenih i dalje ostaje 3.

**Tablica 2.2:** Matrica ekvivalentna prikazu na slici 2.2

| | | | | |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

Jedan od primjera primjene ovog koncepta bio bi: tvrtka zapošljava ljude za određene pozicije u tvrtci. Lijevo neka su prijavnici za posao, a desno neka su pozicije u tvrtci. Direktor tvrtke je ustvrdio temeljen razgovora sa svakim prijavnikom za koje je pozicije tvrtci prijavnik sposoban. Direktor želi zaposliti maksimalan broj ljudi odnosno ispuniti maksimalan broj pozicija u tvrtci prijavnicima. Još neki primjeri bi bili: maksimalno sparivanje otprilike jednako jakih nogometnih momčadi za veliki festival nogometa u subotu, najveće sparivanje daljinskih upravljača i za njih kompatibilnih televizora na skladištu, sparivanje automobilskih motora na skladištu s kompatibilnim automobilima kojima je motor uništen(može zamijeniti riječ motor s akumulator ili automobil s brod) i mnoge druge kreativne primjene.

### 2.2.2. Težinsko sparivanje

Sparivanje u kojem svaki brid ima između nekog elementa lijevog skupa $i$ nekog elementa desnog skupa $j$ asociranu težinu $Wij$.

#### Problem pridruživanja s nejednakim dimenzijama

Neka lijevi skupa ima N elemenata dok desni skup ima M elemenata. Pretpostavimo da vrijedi $NM$ samo transponiramo matricu težina i nastavljamo i sve u nastavku vrijedi). Primjer takvog slučaja je na tablici 2.3.

**Tablica 2.3:** Pravokutna matrica

| | | | | |
|---|---|---|---|---|
| 23 | 68 | 11 | 25 | 97 |
| 39 | 10 | 91 | 3 | 21 |
| 58 | 29 | 75 | 47 | 19 |

Ovo lako možemo svesti na problem pridruživanja. Ideja je da nadopunimo matricu do kvadratne matrice nekom vrijednošću, najlakše nulama. Vidi tablicu 2.4

**Tablica 2.4:** Kvadratna matrica

| | | | | |
|---|---|---|---|---|
| 23 | 68 | 11 | 25 | 97 |
| 39 | 10 | 91 | 3 | 21 |
| 58 | 29 | 75 | 47 | 19 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

Razlog zašto to možemo napraviti je taj da bilo koji vektor rješenja ovog problema na prvih N mjesta ima rješenje originalnog problema budući da odabir zadnjih M-N ne utječe na rješenje jer su svi isti, stoga je odabir samo prvih N bio bitan pri generaciji rješenja.

Na ovo će se češće nailaziti u praksi nego na kvadratni slučaj matrice. Primjer primjene za ovaj pa i kvadratni slučaj je: dostavljači na terenu koji dostavljaju hranu(ili nešto drugo) mušterijama s ciljem minimizacije pređenog puta dostavljača. Primjer je prilično realan, udaljenost $d$ koju bismo uzeli između dostavljača i te mušterije j bi bila $d(i,j) = \min_{k=1}^{R} d(dostavljac[i], restoran[k]) + d(restoran[k], musterija[j])$, gdje je $R$ broj restorana u području. Druge primjene vođene sličnom logikom bi bili: dostava mlijeka od farmi do prodavaonica kombijima, minimalna udaljenost da svi brodovi na moru dođu do neke luke(recimo ribari navečer s ciljem minimizacije potrošenog goriva), pri tom se može modelirati stvar tako da je neka luka predstavljena s više elemenata desnog skupa ukoliko ima kapaciteta za više brodova. Primjena može biti i kao subrutina u nekim drugim matematičkim ili algoritamskim problemima.

### Problem maksimalnog pridruživanja

Sve do sada što smo pričali je bilo definirano za minimizaciju. Problem se može analogno definirati za maksimizaciju. Služeći se ranijim definicijama, rješenje problema maksimalnog pridruživanja je onaj $K \in S$, za koji vrijedi $g(K) \geq g(V), \forall V \in S$.

Problem se svodi na problem minimalnog pridruživanja tako da se svakom bridu pridijeli njegova vrijednost pomnožena s minus jedan, zatim se koristi algoritam za običan problem te uzme rezultat pomnožen s minus 1. S rekonstrukcijom nema problema što ćemo vidjeti u samom algoritmu.

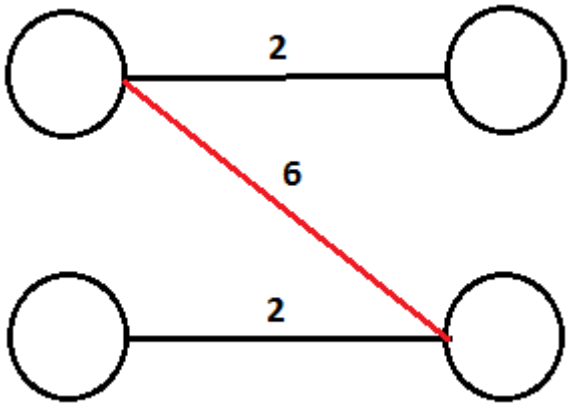
Primjer primjene bio bi redistribucija zaposlenika u tvrtci. Recimo da nakon par mjeseci rada i analize sposobnosti koja je rezultirala procjenom koliko zaposlenik i na poziciji u tvrtci j može generirat prihoda(recimo, kn/h). Direktor zatim odluči preraspodijeliti pozicije u tvrtci zaposlenicima tako da ostvaruje maksimalan prihod.

### Sparivanje maksimalne težine

Susretanje s potpunim bipartitnim grafovima je vrlo rijetko naspram susretanja s nepotpunim bipartitnim grafovima. Međutim nema razloga za paniku jer su potrebne minimalne obzervacija da bismo sveli opći problem na problem pridruživanja.

Za zadani težinski bipartitni graf, samo na mjesto nepostojećih bridova dodamo bridove težine nula. Od rješenja problema pridruživanja nad takvim bridovima uzmemo onda samo one bridove koji stvarno postoje u rekonstrukciji. Rješenje je optimalno zato što nad proširenim skupom bridova je svako drugo rješenje jednako ili lošije od njega, a posljedično je podskup s pravim bridovima bolji od podskupa s pravim bridovima bilo kojeg drugog rješenja jer podskup s nepravim bridovima pridonosi vrijednošću 0.

Valja naglasiti da optimalno rješenje neće uvijek imati najveći mogući kardinalitet sparivanja. Slika 2.3 prikazuje trivijalan primjer takvog scenarija. Također treba naglasiti da inačica zadatka gdje želimo naći minimalno težinsko sparivanje nema smisla budući da bi uvijek iznosila 0. Ne bismo ništa spojili i to bi bilo optimalno, naravno osim inačice s negativnim težinama, ali to se svodi na isto kao i sparivanje maksimalne težine.




**Slika 2.3:** Maksimalno težinsko sparivanje.

Neki primjer primjene vuče na isto kao i prije nabrojane primjene, ali navesti ćemo jednu. Bavimo se autodijelovima. Imamo N automobilskih motora na skladištu i M

automobila kojima je motor nepopravljivo oštećen. Između svakog kompatibilnog para motora $i$ i automobila $j$ postoji procijenjena vrijednost na tržištu ako se motor $i$ ugradi u automobil $j$. Ako (i,j) nisu kompatabilni ugradnja se, naravno ne razmatra. Cilj je maksimizirati ukupnu vrijednost na tržištu popravljenih automobila.

Područje koje se bavi problemom pridruživanja i sličnim problemima te njihovim primjenama zove se operacijska istraživanja(engl. *operations research*).

# 3. Algoritmi za problem pridruživanja

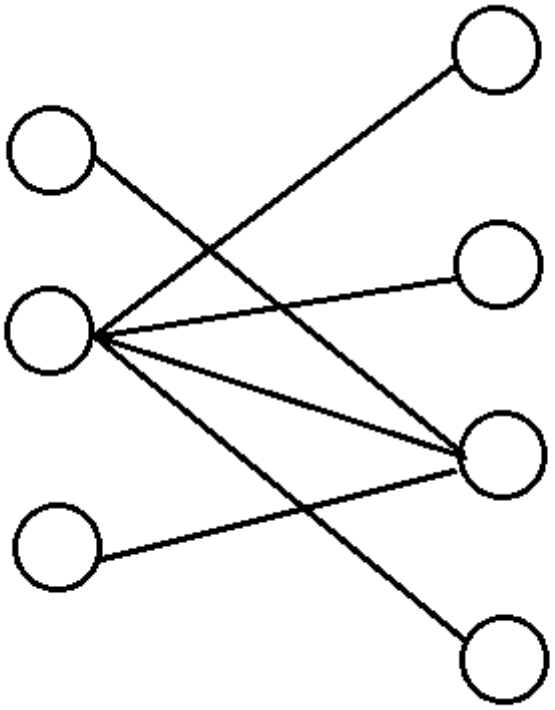
U ovom poglavlju proći ćemo kroz algoritme koji rješavaju problem pridruživanja i njegove posebne slučajeve. Detaljno ćemo opisati i dokazati mađarski algoritam.

## 3.1. Maksimalni protok kroz mrežu

Problem maksimalnog protoka kroz mrežu(engl. *Maxflow*) je usko vezan uz problem pridruživanja. Naime, već spomenuto, problem sparivanja maksimalnog kardinaliteta, koji se da riješiti primjenom maxflow algoritma, je podskup problema pridruživanja. Problem pridruživanja je podskup problema maksimalnog protoka kroz mrežu minimalne cijene(engl. *min cost max flow*), akronimno MCMF.

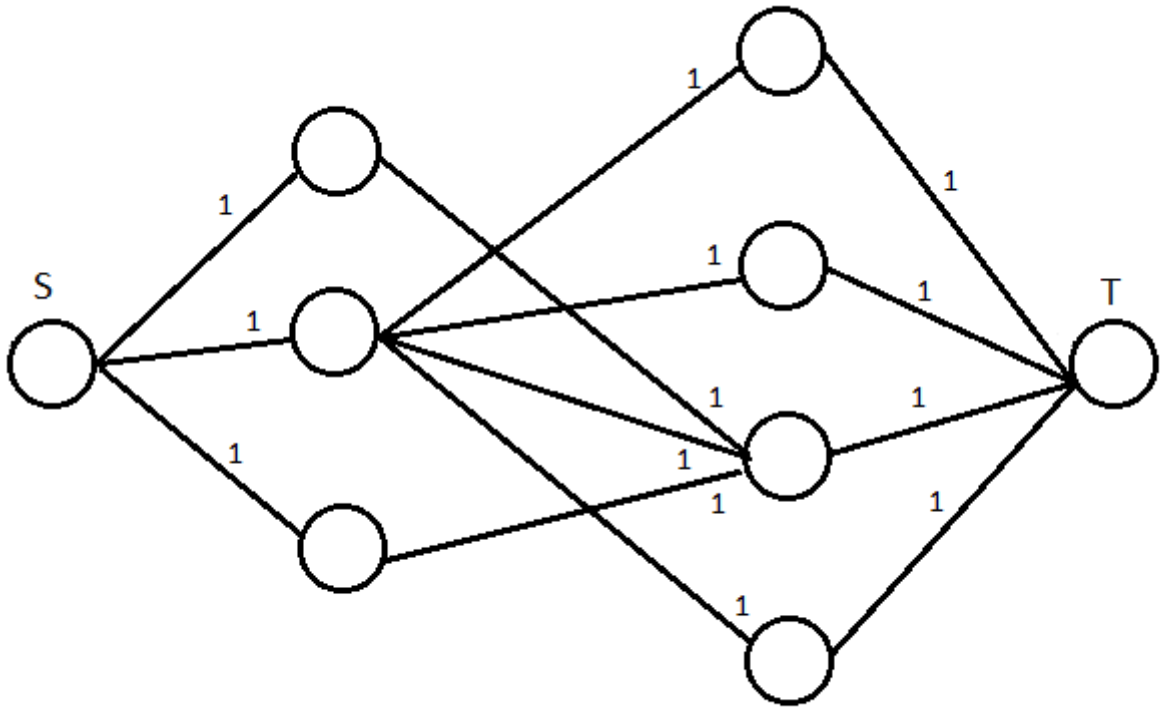
### 3.1.1. Bestežinsko sparivanje

Teoretski ćemo razmotriti korištenje nekog algoritma za maksimalni protok kroz mrežu. Postoje razni algoritmi različitih složenosti: Ford-Fulkerson, Edmonds-Karp, Dinic, Hopcroft-Karp(ovo je doduše specijalizacija baš za bipartitno sparivanje). Uz pretpostavku da znamo neki od navedenih algoritama, možemo problem sparivanja svesti na problem najvećeg protoka. Promotrimo graf na slici 3.1 koji ćemo transformirati.



**Slika 3.1:** Bipartitni graf

Problem svodimo na protok u mreži na sljedeći način: Postojeće veze u grafu pretvorimo u bridove kapaciteta 1. Dodamo fiktivne čvorove $S$ i $T$, poznatiji u engleskoj literaturi kao $source$ i $sink$. $S$ spojimo sa svakim čvorom lijeve skupine bridom kapaciteta 1 te $T$ spojimo sa svakim čvorom desno skupine bridom kapaciteta 1. Transformirani graf je prikazan na slici 3.2.




**Slika 3.2:** Transformirani graf

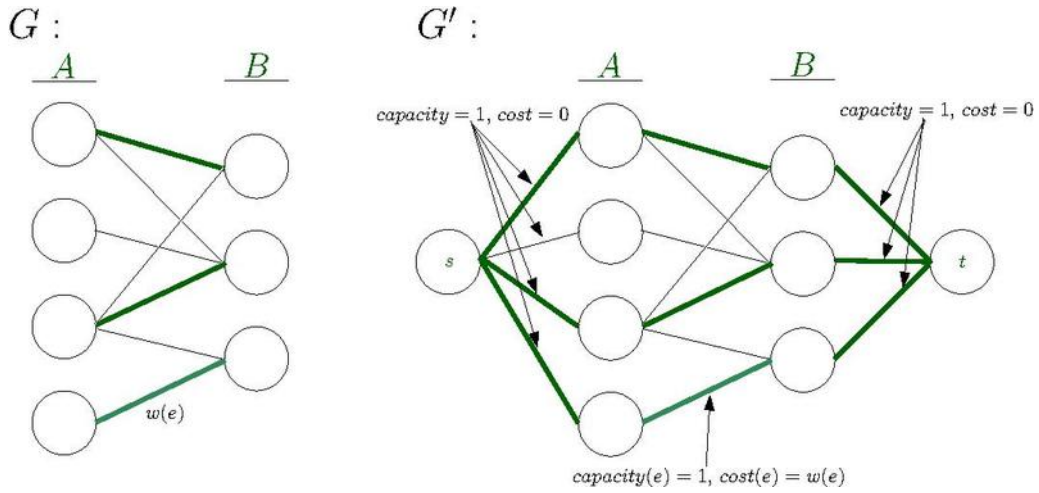
Koristimo recimo, Edmonds-Karp algoritam vremenske složenosti u općem slučaju $O(V * E^2)$ gdje je V broj vrhova, a E broj bridova u grafu. Budući da taj algoritam u svakoj augmentaciji protoka poveća protok za barem 1, složenost na prilagođenom bipartitnom grafu bit će mu $O(V * E)$ , jer će maksimalno V puta(u slučaju potpunog sparivanja) augmentirati protok, za što mu je složenost $O(E)$. Rekonstrukcija protoka u ovoj mreži daje maksimalno sparivanje.

### 3.1.2. Težinsko sparivanje

Ovdje teoretski razmatramo korištenjem nekog algoritma za rješavanje maksimalnog protoka minimalne cijene u svrhu rješavanja problema pridruživanja. Problem maksimalnog protoka minimalne cijene je sljedeći: zadan je graf koji ima izvorište(engl. *source*) i odvod(engl. *sink*), bridovi između dva čvora u tom grafu imaju pridružen par (cijena, kapacitet). Cijena nekog slanja nekog protoka(manjeg od kapaciteta) kroz neki brid $e_i$ je jednaka $protokKroz(e_i) * cijena(e_i)$. Cilj je minimizirati ukupnu cijenu slanja nekog protoka $f$ kroz mrežu od izvorišta do odvoda.

Problem pridruživanja možemo modelirati kao problem protoka slično bestežin-

skom primjeru. Spojimo fiktivni čvor izvorišta lijevim čvorovima u problemu pridruživanja s bridovima $(kapacitet, cijena) = (1, 0)$, isto napravimo s desnima i odvodom. Zatim svaki brid $e_i$ između lijeve i desne strane transformiramo u novi brid istog kapaciteta i cijene 1: $e_i' = (1, w_i)$. Na slici 3.3 je to grafički prikazano.




**Slika 3.3:** Transformacija grafa. Preuzeto 09.06.2022. s:`https://en.wikipedia.org/wiki/Minimum-cost_flow_problem`

Maksimalni protok u problemu pridruživanja bi trebao biti veličine $N$, dakle potpuno sparivanje. MCMF algoritam će od svih maksimalnih sparivanja uzeti ono najmanje cijene. Možemo koristiti, recimo, algoritam [8] vremenske složenosti $O(V^2 Elog(VC))$ gdje je $C$ najveća cijena brida.

## 3.2. Mađarski algoritam

Gradit ćemo se od teoretskih razmatranja prema cjelovitom algoritmu dobre složenosti.

### 3.2.1. Teorija

Ključni mehanizam algoritma je dodjeljivanje težina vrhovima u grafu te držanje invarijante prilikom poboljšavanja putova.

**Definicija 1.** *Definiramo funkciju labeliranja $l : V \mapsto \mathbb{Z}$, , gdje je $V$ skup vrhova u grafu.*

**Definicija 2.** *Neka su $X$ lijevi vrhovi, a $Y$ desni vrhovi u bipartitnom grafu. Kažemo da je funkcija labeliranja $l$ ograničavajuća ako vrijedi $w(x, y) \geq l(x)+l(y), \forall x \in X, y \in Y$. Gdje funkcija $w$ predstavlja težinu između dva čvora.*

Ograničavajuće funkcije osiguravaju da je zbroj bilo kojeg sparivanja u grafu veći ili jednak od zbroja vrijednosti funkcije labeliranja. Drugim riječima, za bilo koje sparivanje $M$ vrijedi $w(M) = \sum_{e \in M} w(e) \geq \sum_{e \in M}(l(e_x) + l(e_y))$. U nastavku slijedi zašto je to korisno.

**Definicija 3.** *Za brid $e$ kažemo da je uzak ako vrijedi $l(e_x) + l(e_y) = w(e)$. Označiti ćemo sa $E_l$ skup uskih bridova.*

Sljedeći teorem govori o postavkama kada je neko sparivanje minimalno. Teorem ne dokazuje egzistenciju takvog namještavanja u svakom bipartitnom grafu niti kako doći do njega. Mađarski algoritam, prikazan nakon teorema pokazuje da je u svakom grafu to moguće namjestiti i algoritam kako to ostvariti.

**Teorem 1.** *Ako je l ograničavajuća i M je potpuno sparivanje koristeći samo bridove iz $E_l$ onda je M minimalno težinsko sparivanje.*

*Dokaz.* Težina sparivanja M je $w(M) = \sum_{e \in M} w(e) = \sum_{e \in M}(l(e_x) + l(e_y)) = \sum_{v \in V} l(v)$. Dakle, vrijedi $w(M) = \sum_{v \in V} l(v)$ odnosno težina sparivanja M je jednaka sumi labela na čvorovima. Promatrajmo bilo koje drugo potpuno sparivanje M'. Vrijedi $w(M') = \sum_{e \in M'} w(e) \geq \sum_{e \in M'}(l(e_x) + l(e_y)) = \sum_{v \in V} l(v) = w(M)$. Odnosno vrijedi $w(M') \geq w(M)$ čime je teorem dokazan. □

Kao što je i najavljeno, sad trebamo pronaći algoritam koji nas uvijek dovodi u situaciju da imamo potpuno sparivanje koristeći samo $E_l$.

### 3.2.2. Opis algoritma

Prvo ćemo idejno bez implementacijskih detalja opisati algoritam. Bez gubitka općenitosti možemo pretpostaviti da su sve težine u grafu pozitivne. Neka $X$ i $Y$ označavaju lijevu i desnu stranu grafa.

**Korak 0**. Inicijalno postavimo labele na svim vrhovima na vrijednost nula, $l(v) = 0, \forall v \in V$. Krećemo se uvijek samo po uskim bridovima. Skup sparenih bridova M je početno prazan.

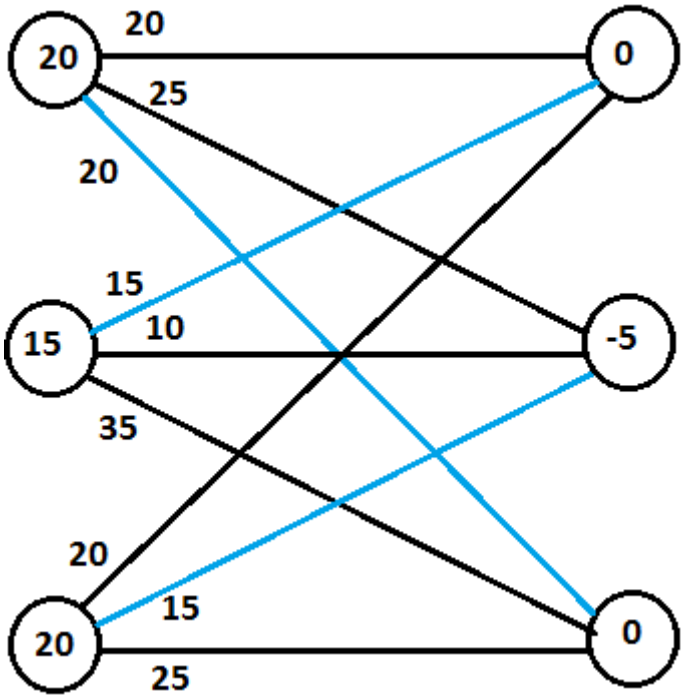
**Korak 1**. Ako je M potpun stajemo. Inače Odaberemo sve nesparene čvorove iz skupa $X$ i stavimo ih u novi skup $T$.

**Korak 2**. Pokrećemo pretraživanje u širinu te ga provodimo sve dok ne naiđemo na nespareni vrh na Y strani. Pritom se krećemo od Y prema X samo po već sparenim bridovima, dok od X prema Y kako po svima po kojima se može. Svaki put do nekog Y čvora je neparne duljine. Svaki čvor na koji naiđemo stavljamo u skup $T$.

**Korak 3**. Ako u skupu $T$ postoji nesparen čvor, iskoristit ćemo ga za povećanje broja sparenih bridova za jedan. Vratit ćemo se unazad po od kud smo došli do tog brida proglašavajući neparne bridove sparenima, a parne nesparenima. Dakle za duljinu povratnog puta n+1 uzimamo čvorove n+1,n-1...3,1 kao sparene. Time smo povećali kardinalitet od $M$ za 1 te idemo nazad na korak 1.

**Korak 4**. Inače, ako u skupu $T$ nije nađen nesparen čvor: Neka je $L = T \setminus Y$ i $R = Y \setminus T$. Tražimo najmanju razliku $d = min_{x,y}(w(x,y) - (l(x) + l(y)), x \in L, y \in R$. Zatim ćemo povećati sve labele iz $L$ za d i smanjiti sve labele iz $T \setminus X$ za d; $l(x) = l(x) + d, \forall x \in L$ te $l(y) = l(y) - d, \forall y \in T \setminus X$. Pri tome se $E_l$ po definiciji mijenja nadodavanjem barem jednog brida. Bridovi koji su prije bili uski i dalje ostaju uski. Važna stvar za primjetiti ovdje je da pri nijednom ograničenost funkcije $l$ ostane invarijantna, nigdje nismo učinili ovom operacijom da vrijedi $w(e) < l(e_x) + l(e_y)$. Ponovno pokreni korak 1.

U koraku 4 moguće je da neki bridovi iz desnog dijela $T$ strukture povezani s lijevim čvorovima koji nisu u $T$ otpadnu smanjivanjem nekog l(y),ali to nam nije bitno. Svaki put kad se vratimo u korak 1 znači da smo povećali kardinalitet od M za točno jedan. Složenost ovakvog pristupa bila bi $O(n^4)$ gdje je n broj vrhova na jednoj od strana. Slika 3.4 prikazuje završne postavke algoritma na jednom primjeru od prije.




**Slika 3.4:** Izgled grafa pri završetku mađarskog algoritma.

Algoritam se može ubrzati na $O(n^3)$ ako se modificiraju koraci 1,2 i 4. Ideja je u tome da ne skačemo potencijalno n puta u korak 1 iz koraka 4 ponavljajući BFS koji iziskuje $O(n^2)$ iteracija. Kako bismo to postigli stvaramo pomoćni niz $desno[i]$ u kojem čuvamo koje je još povećanje težina na labelama $T \setminus Y$ potrebno kako bi neki čvor iz $Y \setminus T$ postao pristupačan. Inicijalno sve vrijednosti desno[i] postavimo na "beskonačno", te zatim tijekom BFS-a ažuriramo te vrijednosti i na kraju gledamo samo one van skupa T i sve ih modificiramo. Poboljšani algoritam slijedi u nastavku:

**Korak 0**. Inicijalno postavimo labele na svim vrhovima na vrijednost nula, $l(v) = 0, \forall v \in V$. Krećemo se uvijek samo po uskim bridovima. Skup sparenih bridova M je početno prazan.

**Poboljšani korak 1**. Ako je M potpun stajemo. Inače Odaberemo sve nesparene čvorove iz skupa $X$ i stavimo ih u novi skup $T$. Stvorimo niz duljine n $desno$ i postavimo mu sve vrijednosti na inf, $desno[i] = inf, \forall i$.

**Poboljšani korak 2**. Pokrećemo pretraživanje u širinu te ga provodimo sve dok ne naiđemo na nespareni vrh na Y strani. Pritom se krećemo od Y prema X samo po već sparenim bridovima, dok od X prema Y kako po svima po kojima se može. Svaki put do nekog Y čvora je neparne duljine. Svaki čvor na koji naiđemo stavljamo u skup $T$. Prilikom obrade svakog čvora s lijeve strane gledamo veze između njega i čvorova desne strane kako bismo smanjili desno[i], konkretno ako smo na lijevom čvoru x onda $desno[i] = min(desno[i], w(x,i) - (l[x] + l[i])), \forall i$.

**Korak 3**. Ako u skupu $T$ postoji nesparen čvor, iskoristit ćemo ga za povećanje broja sparenih bridova za jedan. Vratit ćemo se unazad po od kud smo došli do tog brida proglašavajući neparne bridove sparenima, a parne nesparenima. Dakle za duljinu povratnog puta n+1 uzimamo čvorove n+1,n-1...3,1 kao sparene. Time smo povećali kardinalitet od $M$ za 1 te idemo nazad na korak 1.

**Poboljšani korak 4**. Inače, ako u skupu $T$ nije nađen nesparen čvor: Neka je $L = T \setminus Y$ i $R = Y \setminus T$. Tražimo najmanju razliku $d = min_y(desno[y]), y \in R$. Zatim ćemo povećati sve labele iz $L$ za d i smanjiti sve labele iz $T \setminus X$ za d; $l(x) = l(x) + d, \forall x \in L$ te $l(y) = l(y) - d, \forall y \in T \setminus X$. Pri tome se $E_l$ po definiciji mijenja nadodavanjem barem jednog brida. Bridovi koji su prije bili uski i dalje ostaju uski. Važna stvar za primijetiti ovdje je da pri nijednom ograničenost funkcije $l$ ostane invarijantna, nigdje nismo učinili ovom operacijom da vrijedi $w(e) < l(e_x) + l(e_y)$. Umetni u red(engl. *queue*) za pretragu sve $y \in R$ koji imaju vrijednost $desno[y] = 0$. Pokreni ponovno korak 2.

# 4. Vrednovanje programskih ostvarenja

U sklopu rada programski su ostvareni mađarski algoritam i Edmonds-Karp algoritam u C++u.

## 4.1. Sparivanje maksimalnog kardinaliteta

Uspoređivat ćemo performanse Edmonds-Karp algoritma i mađarskog algoritma na sparivanju maksimalnog kardinaliteta. Autorova pretpostavka je da bi Edmonds-Karp trebao biti brži jer mađarski algoritam ima veliku vremensku konstantu zbog svoje kompliciranosti izvedbe. Također, Edmonds-Karp bi se trebao vrtjeti u $O(velicinaSparivanja * m)$, dok bi mađarski algoritam trebao $O(n * m)$, odnosno n puta prijeći m iteracija bez obzira na to što je rješenje manje od n. Tablica 4.1 prikazuje rezultate.

**Tablica 4.1:** Tablica prosječnih vremena izvođenja u sekundama.

| N | Edmonds-Karp | mađarski | $\frac{t_m a}{t_E K}$ |
|---|---|---|---|
| 20 | 0 | 0.00536 | - |
| 50 | 0.00009 | 0.00736 | 81.778 |
| 75 | 0.00027 | 0.01218 | 45.111 |
| 100 | 0.00009 | 0.02109 | 234.333 |
| 125 | 0.00091 | 0.03582 | 39.363 |
| 150 | 0.00100 | 0.05509 | 55.09 |
| 200 | 0.00200 | 0.12055 | 60.275 |
| 300 | 0.00473 | 0.37657 | 79.613 |
| 400 | 0.00791 | 0.87996 | 111.247 |
| 500 | 0.01591 | 1.80556 | 113.486 |
| 750 | 0.03036 | 5.99452 | 197.447 |
| 1000 | 0.05500 | 14.07835 | 255.970 |
| 1250 | 0.07428 | 27.41975 | 369.140 |
| 1500 | 0.12110 | 47.18579 | 389.643 |
| 1750 | 0.15955 | 75.89261 | 475.667 |
| 2000 | 0.21692 | 113.86469 | 524.916 |

Prvotna pretpostavka je bila točna, Edmonds Karp pokazuje za 3 reda veličine bolje performanse. Mađarski algoritam se ponaša onako kako bi se i ponašao algoritam $O(n^3)$ s visokom konstantom.

## 4.2. Problem pridruživanja

Performanse našeg ostvarenja mađarskog algoritma. Primjeri su pseudoslučajno generirani korištenjem C++ funkcije rand() i vremenskim sjemenom (srand(time(NULL)). Tablica 4.2 prikazuje performanse $O(n^3)$ mađarskog algoritma.

**Tablica 4.2:** Tablica vremena izvođenja u sekundama.

| N | minimalno | medijalno | maksimalno | prosječno | $(\frac{n_i}{n_{i-1}})^3$ | $\frac{prosjek_i}{prosjek_{i-1}}$ |
|---|---|---|---|---|---|---|
| 5 | 0.005 | 0.006 | 0.019 | 0.007 | - | - |
| 10 | 0.005 | 0.006 | 0.006 | 0.006 | 8.000 | 0.818 |
| 20 | 0.006 | 0.006 | 0.007 | 0.006 | 8.000 | 1.063 |
| 50 | 0.006 | 0.007 | 0.008 | 0.007 | 15.625 | 1.164 |
| 75 | 0.008 | 0.009 | 0.011 | 0.009 | 3.375 | 1.282 |
| 100 | 0.012 | 0.013 | 0.013 | 0.013 | 2.370 | 1.410 |
| 125 | 0.016 | 0.018 | 0.019 | 0.018 | 1.953 | 1.390 |
| 150 | 0.023 | 0.024 | 0.025 | 0.024 | 1.728 | 1.362 |
| 200 | 0.043 | 0.046 | 0.048 | 0.046 | 2.370 | 1.891 |
| 300 | 0.122 | 0.126 | 0.128 | 0.126 | 3.375 | 2.741 |
| 400 | 0.270 | 0.276 | 0.295 | 0.279 | 2.370 | 2.217 |
| 500 | 0.536 | 0.546 | 0.565 | 0.548 | 1.953 | 1.963 |
| 750 | 1.888 | 1.913 | 2.055 | 1.944 | 3.375 | 3.551 |
| 1000 | 4.542 | 4.605 | 5.116 | 4.674 | 2.370 | 2.404 |
| 1250 | 8.320 | 8.872 | 9.662 | 8.872 | 1.953 | 1.898 |
| 1500 | 13.900 | 15.024 | 15.942 | 14.952 | 1.728 | 1.685 |
| 1750 | 22.157 | 22.469 | 23.264 | 22.554 | 1.588 | 1.508 |
| 2000 | 32.769 | 33.381 | 33.735 | 33.245 | 1.493 | 1.474 |

Pod pretpostavkom da je složenost koda neka konstanta put složenost $konst * O(n^3)$, kvocijent $(\frac{n_i}{n_{i-1}})^3$ za dvije različite vrijednosti od n bi trebao odgovarati vrijednosti $\frac{prosjek_i}{prosjek_{i-1}}$. U zadnja dva stupca tablice vidimo da se to poprilično vjerodostojno odražava. Razlog zašto je mađarski algoritam brži na problemu pridruživanja nego na sparivanju maksimalnog kardinaliteta može ležati u tome da ovdje imamo veće težinske skokove pri povećavanju labela.

Usporedit ćemo performanse jednog MCMF algoritma s mađarskim. Asimptotska složenost korištenog algoritma [2] je $O(n^3 * m)$ što bi u slučaju problema pridruživanja značilo $O(n^5)$. Nije pružena analiza asimptotske složenosti algoritma na bipartitnom grafu, ne može se izostaviti mogućnost da je dokazljivo manja. Prikaz na tablici 4.3.

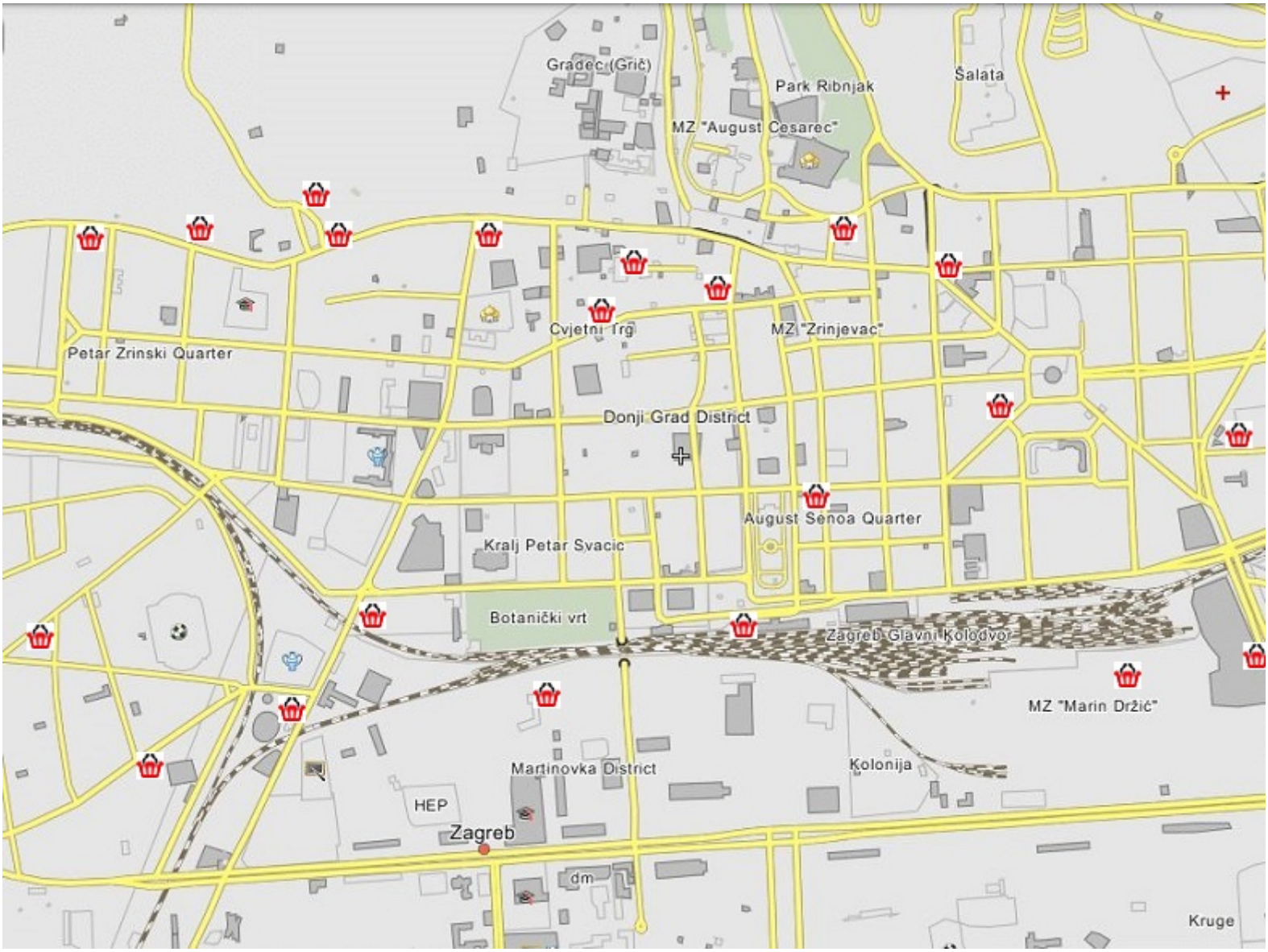
**Tablica 4.3:** Usporedba prosječnih vremena dva algoritma u sekundama.

| N | mađarski | MCMF | $\frac{t_m cmf}{t_h un}$ |
|---|---|---|---|
| 20 | 0.006 | 0.000 | - |
| 50 | 0.007 | 0.004 | 0.571 |
| 75 | 0.009 | 0.012 | 1.333 |
| 100 | 0.013 | 0.027 | 2.077 |
| 125 | 0.018 | 0.048 | 2.667 |
| 150 | 0.024 | 0.086 | 3.583 |
| 200 | 0.046 | 0.239 | 5.195 |
| 300 | 0.126 | 0.900 | 7.143 |
| 400 | 0.279 | 2.157 | 7.731 |
| 500 | 0.548 | 4.221 | 7.703 |
| 750 | 1.944 | 14.117 | 7.262 |
| 1000 | 4.674 | 32.706 | 6.997 |
| 1250 | 8.872 | 68.280 | 7.696 |
| 1500 | 14.952 | 109.56 | 7.327 |
| 1750 | 22.554 | 169.838 | 7.530 |
| 2000 | 33.245 | 254.270 | 7.648 |
| 3500 | 3min 12s | 23min 5s | 7.207 |
| 5000 | 10 min | 1 sat 5 minuta | 6.475 |

Rezultati u tablici sugeriraju da su algoritmi temeljeni na protoku mnogo brži od njihove asimptotske složenosti što je razumna pretpostavka te uvriježeno i izmjereno ponašanje za mnoge algoritme temeljene na protoku. Mađarski algoritam se ponaša poprilično brže od MCMF. Postoje pseudo-polinomni algoritmi[5] asimptotske složenost $O(\sqrt{n} * m * log(nC))$, gdje C predstavlja najveću cijenu na nekom bridu. Takvi algoritmi imaju potencijala biti bolji od mađarskog algoritma za sve praktične probleme.

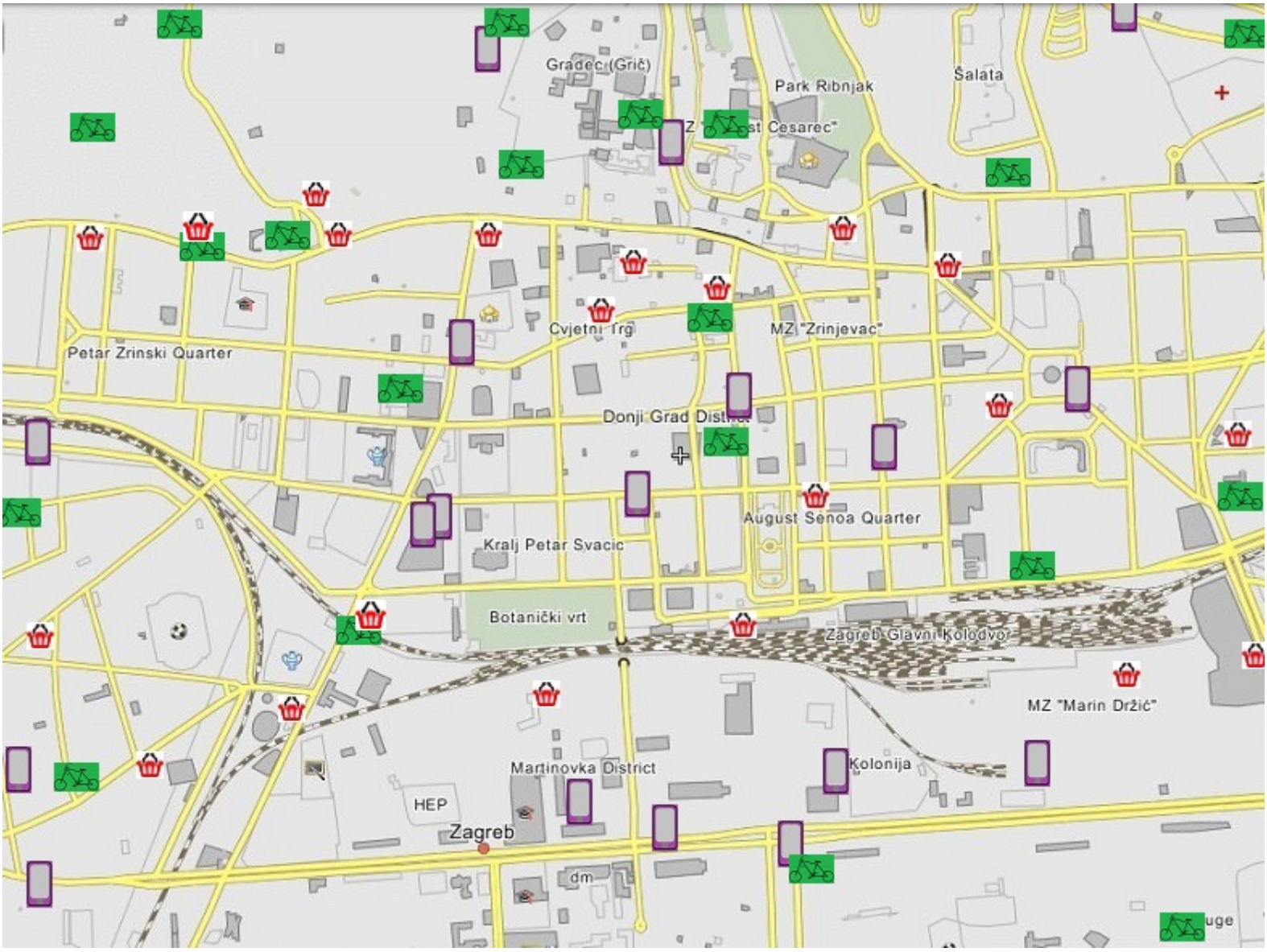
## 4.3. Primjena mađarskog algoritma u dostavi

Primijeniti ćemo ostvareni program kako bismo minimizirali troškove dostave jednog poznatog hrvatskog lanca trgovina. Pretpostavimo da Konzum uvede dostavljače posvećene dostavi na biciklima. Može se pretpostaviti i da dostavljači imaju privilegirane ulaze i izlaze u trgovinama preko nečeg poput kartica, drugim riječima ne moraju čekati dugo na blagajnama. Promatrajmo slučaj grada Zagreba zajedno s Konzumovim pozicijama trgovina na slici 4.1. Pozicije trgovina su stvarne pozicije trgovina Konzum u gradu Zagrebu.




**Slika 4.1:** Karta Zagreba s naznačenim trgovinama.

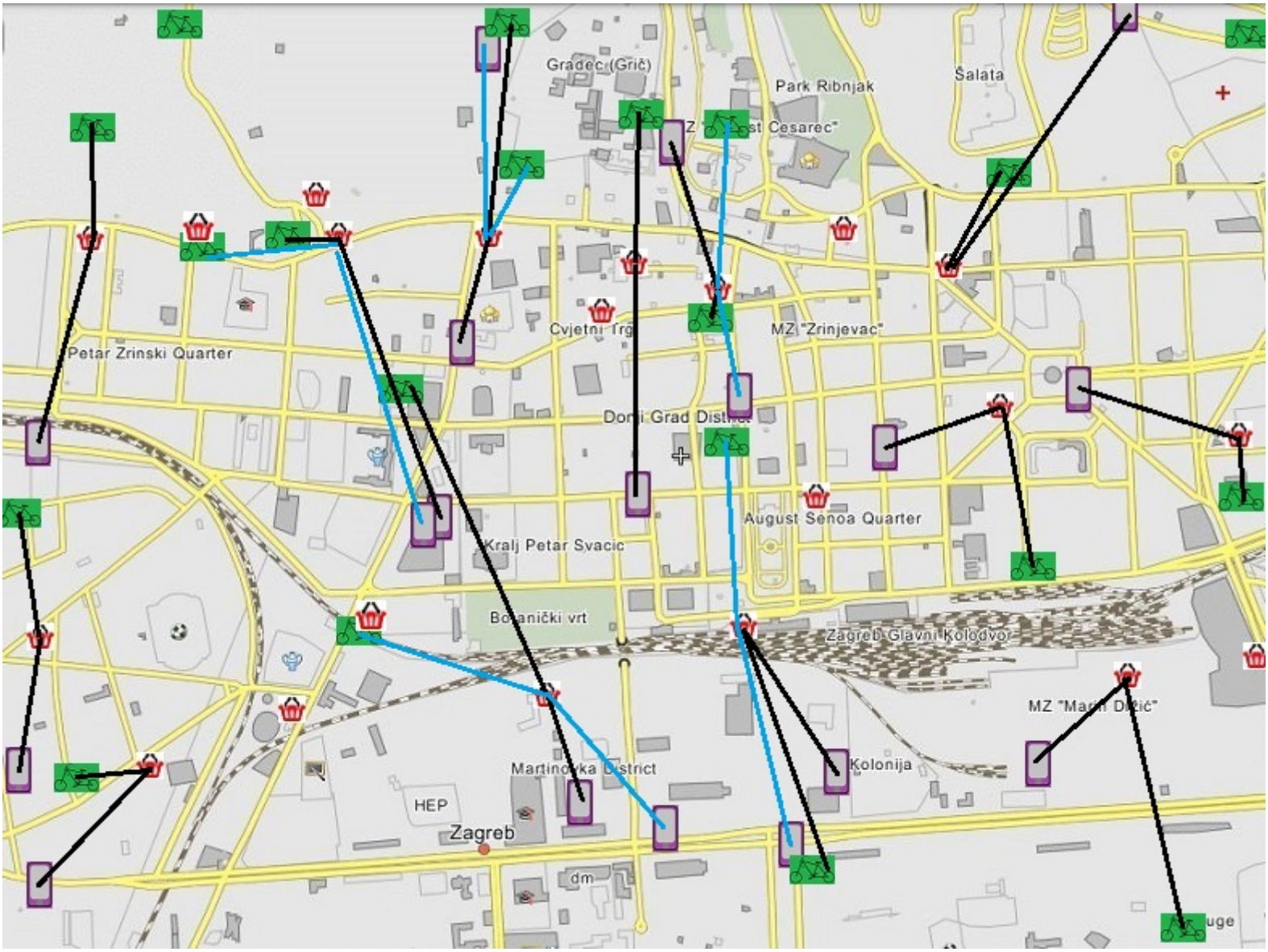
Pretpostavimo sada da imamo svoje dostavljače negdje na terenu, označene zelenim biciklom. Također imamo i narudžbe od koje treba ispuniti, označene ljubičastim mobitelom. Shema dodjeljivanja narudžbi nije da odmah dodjeljujemo dostavljača narudžbi već čekamo 5 minuta da se narudžbe nakupe kako bismo ih optimalno rasporedili te da se dostavljači malo odmore. Prikaz na slici 4.2.




**Slika 4.2:** Karta Zagreba s naznačenim dostavljačima, narudžbama i trgovinama.

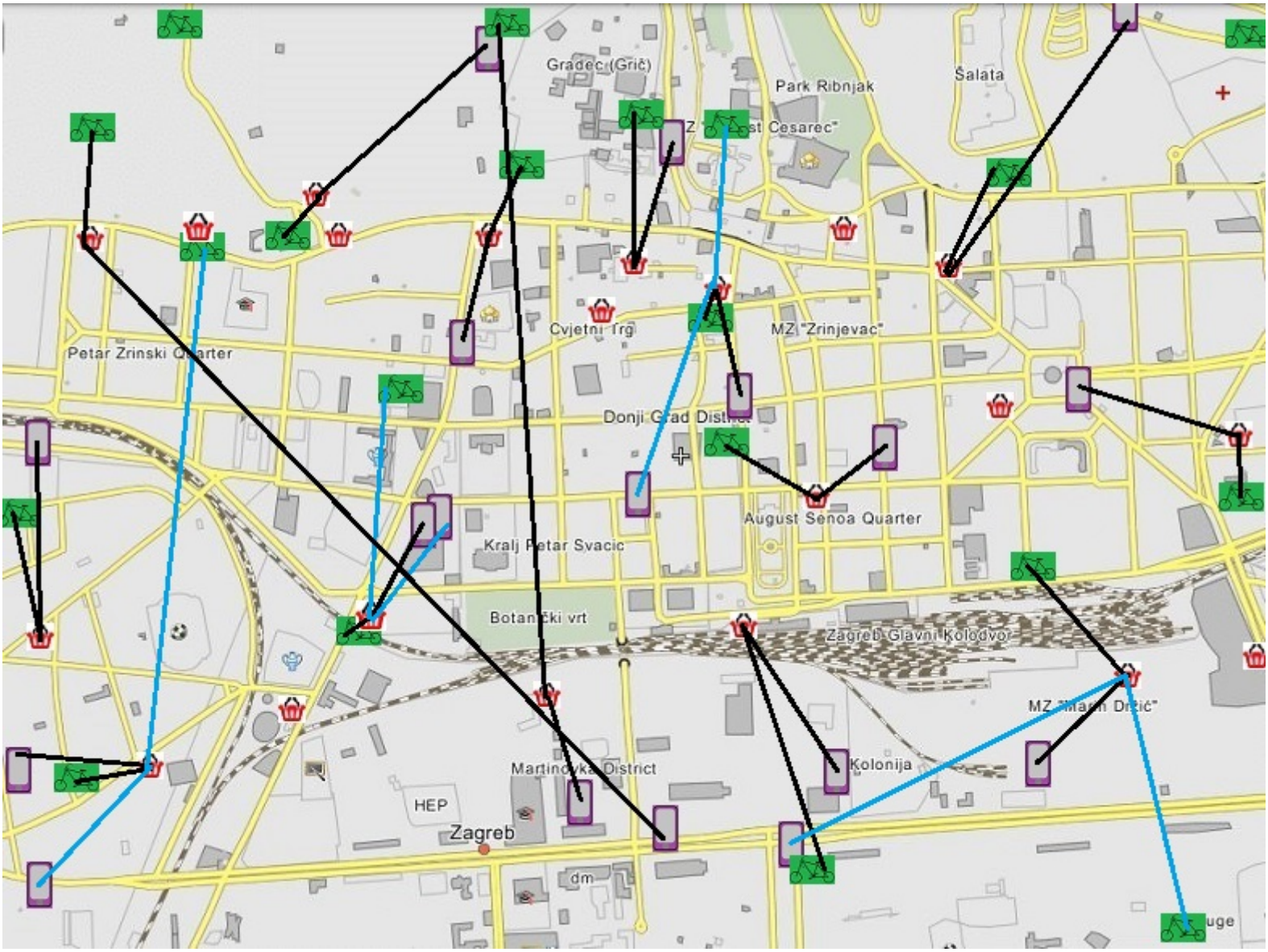
Cilj bi sad bio spariti dostavljače na biciklima s narudžbama tako da dostavljači pređu minimalan put kako bi se čim manje trošili bicikli. Implicitno time se i manje "troše" dostavljači u vidu posljedica dugoročnog rada na zdravlje, moguće ozljede i slično. Pretpostavka je da svaka trgovina na inventaru ima sve što treba, pa je zato dovoljno obići samo jednu trgovinu prilikom puta do narudžbe.

Problem pretvaramo u problem pridruživanja na sljedeći način: gledamo narudžbe kao lijevi dio bipartitnog grafa te gledamo dostavljače kao desni dio. Udaljenost između narudžbe $i$ i dostavljača $j$ je određeno kao $d(i,j) = \min_k d(narudzba[i], konzum[k]) + d(konzum[k], narudzba[j])$. Kao udaljenost koristit ćemo Manhattan udaljenost dvije točke zbog jednostavnosti, a i nije loša aproksimacija prave udaljenosti. Broj narudžba i dostavljača neće uvijek biti isti no program s već prije opisanim varijacijama može to riješiti. U našem primjeru je 18 narudžbi i 20 dostavljača stoga 2 dostavljača ostanu nedodijeljena. Optimalno sparivanje prikazano je na slici 4.3.




**Slika 4.3:** Optimalno sparivanje.

Uz aproksimaciju da je 1 pixel 3.75 metara, ukupno pređena udaljenost ispada 16.95km. Koristeći pohlepni pristup poput *spari svaki put brid najmanje težine* dobivamo 19.83 km. to je poboljšanje od 14.45% te vjerojatno mnogo veće od kompanije koja ne koristi nikakav algoritam dodjeljivanja, kako god to funkcioniralo. Na drugim primjerima položaja narudžbi i dostavljača(Konzum ostaje fiksan) mogu se isprobati još neki pohlepni algoritmi. Primjeri još nekih pohlepnih algoritama bi bili: *spajaj najbližu narudžbu i dostavljača*, zatim *spajaj slijedno narudžbe s najmanjim dist-om još dostupnim*, itd. U pravilu se pokazuje da optimalno rješenje pokazuje 5-10% bolje rezultate od pohlepnih pristupa. Slika 4.4 prikazuje prije navedeno povezivanje dugo 19.83 km dobiveno pohlepnim algoritmom *spari svaki put brid najmanje težine*.




**Slika 4.4:** Pohlepno sparivanje.

Vremenski gledano, mađarski algoritam je sasvim primjenjiv na primjer Zagreba koji ima možda par stotina dostavljača po tvrtci, to bi se izračunalo unutar sekunde. U slučaju mega gradova koji imaju više tisuća dostavljača(više od 2000), mađarska metoda bi se mogla svesti na ne optimalan, ali i dalje vrlo dobar kvalitetan dobar pristup. Podijelili bismo grad na četvrti ili nešto slično. Ako smo podijelili grad na $k$ četvrti onda je složenost otprilike $O(k * (\frac{n}{k})^3) = O(\frac{n^3}{k^2})$, $k$ se uzme takav da, recimo računanje traje manje od minute u prosjeku.

# 5. Zaključak

Ostvarili smo pregled problema pridruživanja. Opisali smo i dokazali adekvatan i efektivan algoritam za njegovo rješavanje. Postoje mnogi problemi i primjene koji se relativno lagano modeliraju u problem pridruživanja. Zbog nepotpunosti i razbacanosti literature na ovu temu na Internetu vjerujem da će ovaj dokument koristiti bilo kome tko želi proučiti tematiku i njene varijacije kao i razumjeti zašto mađarska metoda ispravno radi. Trenutno stanje razvoja(engl. *state-of-the-art*) na MCMF(Minimum cost max flow) problemima[6] teži prema tome da bi moglo pretećи asimptotsku vremensku $O(n^3)$ složenost mađarskog algoritma.

# LITERATURA


[1] https://hsm.stackexchange.com/questions/11128/did-jacobi-invent-the-hungarian-algorithm-for-the-assignment-problem-over-a-cent.

[2] http://e-maxx.ru/algo/min_cost_flow, 2008.

[3] https://www.spoj.com/problems/MILPATR/, 2011.

[4] G: Cordon bleu. https://swerc.eu/2017/problems/, 2017.

[5] Andrew P Goldberg et al. An efficient cost scaling algorithm for the assignment problem. *Mathematical Programming*, 71(2):153–177, 1995.

[6] Li Chen et al. Maximum flow and minimum-cost flow in almost-linear time. https://arxiv.org/abs/2203.00671, 2022.

[7] Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83—97, 1955.

[8] James B Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 78(2):109—-129, 1997.

## Problem pridruživanja: rješenja i primjene


### Sažetak

U ovom radu predstavljamo problem pridruživanja i njegove inačice. Opisana je povijesna pozadina razvoja matematičkih koncepata i algoritama koji vode rješenju problema. Dani su primjeri primjene inačica problema. Proučavamo algoritme za rješavanja protoka u svrhu rješavanja problema i njegovih specijalnih slučajeva. Detaljno je opisan i dokazan mađarski algoritam vremenske složenost $O(n^3)$. Rješenja su implementirana u programskom jeziku C++. Prikazano je i uspoređeno vrijeme izvođenja mađarskog, MCMF(engl. *min cost max flow*) i Edmonds-Karp algoritma na pseudo slučajno generiranim primjerima. Pružena je analiza realne primjene algoritma. Osvrćemo na trenutno stanje razvoja algoritama u ovom području.



**Ključne riječi:** maksimalni protok kroz mrežu, bipartitno spajanje, teorija grafova, mađarski algoritam, dostava


## Assignment Problem: Solutions and Applications


### Abstract

In this work, we present the assignment problem and its variants. We describe historical background of the development of mathematical concepts and algorithms that led to the solution of the problem. Application examples of problem variants are given. We study flow-solving algorithms to solve assignment problem and its special cases. Hungarian algorithm of complexity $O(n^3)$ is described in detail and proven. The solutions are implemented in the C++ programming language. Running times of Hungarian, MCMF(min cost max flow) and Edmonds-Karp algorithms are presented and compared on pseudorandom generated examples. Analysis of real-life application algorithm is given. We take a look at current state-of-the-art algorithms in this area.



**Keywords:** Maxflow, bipartite matching, graph theory. Hungarian algorithm, delivery