Problem raspoređivanja (asignacije)

Motivacija i postavka problema

Problem raspoređivanja (ili ***asignacije***) je jedan vrlo značajan problem, koji po svojoj postavci zapravo spada u oblast ***kombinatorne optimizacije***. Međutim, pokazuje se da se, pomalo neočekivano, ovaj problem može posmatrati kao *specijalan slučaj transportnog problema*, a samim tim i *specijalan slučaj općih zadataka linearnog programiranja*. Stoga su, po svojoj *strukturi*, a donekle i *načinu rješavanja*, ovi problemi *slični problemima transporta* (mada ćemo uskoro vidjeti da rješavanje ovih problema metodima razvijenim za probleme transporta, a pogotovo metodima za rješavanje općih problema linearnog programiranja, *nije nimalo dobra ideja*).

Sam problem raspoređivanja se dosta općenito može predstaviti na sljedeći način. Na raspolaganju nam je *m izvršilaca* odnosno *radnika*, pri čemu se pod ovim pojmom podrazumijeva *neko* ili *nešto* što može da obavi ili proizvede neku aktivnost (npr. radnik, student, takmičar, prodavnica, novčani ulog, itd.). Također nam je na raspolaganju *n mjesta* (odnosno *radnih zadataka*) na kojima se mogu realizirati te aktivnosti (npr. poslovi, mašine, ispiti, radna mjesta, sportske discipline, pogoni, lokacije, itd.). Pri tome, *ma koji izvršilac* može da bude raspoređen *na samo jedno mjesto*. Isto tako, svakom mjestu može biti dodijeljen *samo jedan izvršilac*. Svaki *konkretan raspored* proizvodi određeni *efekat* (povećanje vrijednosti, ili povećanje troška), koji je *poznat*. Konkretno, ako se izvršilac I_i, $i = 1 \,..\, m$ rasporedi na mjesto M_j, $j = 1 \,..\, n$ time se ostvaruje *efekat* (*dobit* ili *trošak*) $c_{i,j}$, $i = 1 \,..\, m$, $j = 1 \,..\, n$ pri čemu se smatra da su veličine $c_{i,j}$ poznate. Potrebno je pronaći *takav raspored zadanih izvršilaca na zadana mjesta* koji će dati *optimalni efekat* (maksimalnu dobit ili minimalne troškove), poštujući postavljena ograničenja.

Teoretski, problem raspoređivanja bi se mogao riješiti *ispitivanjem svih mogućih razmještaja* i biranjem onog razmještaja koji daje *najbolji efekat*. Problem je što broj svih mogućih razmještaja *iznimno brzo raste* sa porastom m i n. Na primjer, neka je $m = n$. Prvog izvršioca možemo rasporediti na jedno od n različitih mjesta. Nakon toga drugog izvršioca možemo rasporediti na jedno od $n-1$ preostalih mjesta (jer ne smijemo dva izvršioca rasporediti na isto mjesto), trećeg izvršioca na jedno od $n-2$ preostalih mjesta, i tako dalje sve do posljednjeg izvršioca kojem ostaje na raspolaganju samo jedno mjesto. Slijedi da je ukupan broj razmještaja $n \cdot (n-1) \cdot (n-2) \cdot 3 \cdot 2 \cdot 1 = n!$. Tako, već za $n = 10$ imamo $10! = 3628800$ mogućih razmještaja, a za $n = 20$ imamo $20! = 2432902008176640000$ mogućih razmještaja. Slijedi da je ovakav pristup *posve neprimjenljiv* za iole veće zadatke raspoređivanja. Računaru koji je u stanju da analizira *milijardu* razmještaja u jednoj sekundi, trebalo bi *nešto više od 77 godina* da na ovaj način riješi problem raspoređivanja već za $n = 20$! Slijedi da nam je potreban *bolji pristup*.

Prvi korak ka sagledavanju problema je *formiranje njegovog matematičkog modela*. Prvo pitanje koje se nameće je *šta su ovdje uopće nepoznate veličine koje trebamo odrediti*. Jedna ideja je da uvedemo za svakog izvršioca I_i, $i = 1 \,..\, m$ po jednu promjenljivu, recimo x_i, $i = 1 \,..\, m$ čija bi vrijednost mogla biti samo cijeli broj u opsegu od 1 do n, pri čemu bi $x_i = k$ značilo da se i-ti izvršilac treba rasporediti na k-to mjesto. Međutim, ovakav pristup, mada djeluje prirodan na prvi pogled, *nije lako pretočiti u pogodan i upotrebljiv matematički model*.

Pri modeliranju *kombinatornih problema*, jedan drugi pristup se pokazuje kao mnogo pogodniji. Primijetimo da za svakog mogućeg izvršioca I_i, $i = 1 \,..\, m$ i svako moguće mjesto M_j, $j = 1 \,..\, n$ postoji *jedna i samo jedna od sljedeće dvije mogućnosti*: ili će izvršilac I_i *biti raspoređen* na mjesto M_j, ili izvršilac I_i *neće biti raspoređen* na mjesto M_j. Uvešćemo stoga promjenljive $x_{i,j}$, $i = 1 \,..\, m$, $j = 1 \,..\, n$ sa sljedećim značenjem:

$$x_{i,j} = \begin{cases} 1, & \text{ako je izvršilac } \mathrm{I}_i \textit{ raspoređen} \text{ na mjesto } \mathrm{M}_j \\ 0, & \text{ako izvršilac } \mathrm{I}_i \textit{ nije raspoređen} \text{ na mjesto } \mathrm{M}_j \end{cases}$$

Ovaj pristup omogućava da veoma lako izrazimo funkciju cilja. Zaista, ukoliko je i-ti izvršilac raspoređen na j-to mjesto, time se ostvaruje efekat $c_{i,j}$, dok ukoliko to nije slučaj, time se ne ostvaruje *nikakav efekat* (tj. efekat 0). Obje situacije možemo objediniti tako što ćemo reći da se ostvaruje ukupan efekat $c_{i,j}\, x_{i,j}$ (zaista, za $x_{i,j} = 1$, ovo je $c_{i,j}$, a za $x_{i,j} = 0$, ovo je 0). Stoga je ukupan efekat koji se ostvaruje od *svih raspoređivanja* jednak

$$\begin{aligned} Z = {} & c_{1,1}\, x_{1,1} + c_{1,2}\, x_{1,2} + \ldots + c_{1,n}\, x_{1,n} + \\ & + c_{2,1}\, x_{2,1} + c_{2,2}\, x_{2,2} + \ldots + c_{2,n}\, x_{2,n} + \\ & \quad \ldots \\ & + c_{m,1}\, x_{m,1} + c_{m,2}\, x_{m,2} + \ldots + c_{m,n}\, x_{m,n} \end{aligned}$$

i ovo je funkcija cilja koju treba optimizirati (minimizirati ili maksimizirati). Očigledno se radi o *istom obliku funkcije cilja* kao kod transportnog problema.

Razmotrimo sada i ograničenja. Prvo ćemo posmatrati slučaj kada je $m = n$. U ovom slučaju je očigledno moguće *svakog izvršioca rasporediti na neko mjesto* (jer ima dovoljno mjesta), i na svako mjesto je moguće *rasporediti nekog izvršioca* (jer ima dovoljno izvršilaca). Stoga činjenicu da će *i*-ti izvršilac biti raspoređen na tačno jedno mjesto možemo iskazati kao

$$x_{i,1} + x_{i,2} + \ldots + x_{i,n} = 1$$

Zaista, kako sve promjenljive $x_{i,j}$ mogu biti isključivo 0 ili 1, zbir sa lijeve strane jednak je jedinici ako i samo ako je *tačno jedna* od promjenljivih sa lijeve strane jednaka jedinici. Ovakva ograničenja se mogu napisati za sve izvršioce I_i, $i = 1 \,..\, m$.

Koristeći analogno rezonovanje, uvjet da će na *j*-to mjesto biti raspoređen tačno jedan izvršilac možemo iskazati u obliku

$$x_{1,j} + x_{2,j} + \ldots + x_{m,j} = 1$$

Ista ovakva ograničenja se mogu napisati za sva mjesta M_j, $j = 1 \,..\, n$. Zajedno sa ograničenjima da sve promjenljive mogu uzimati samo vrijednosti 0 ili 1, ovo daje sljedeći matematski model problema raspoređivanja, uz pretpostavku da razmatramo problem minimizacije (u slučaju maksimizacije, samo se mijenja "min" u "max", dok sva ograničenja ostaju ista):

$$\begin{aligned}\arg\min Z = {} & c_{1,1}x_{1,1} + c_{1,2}x_{1,2} + \ldots + c_{1,n}x_{1,n} + \\ & + c_{2,1}x_{2,1} + c_{2,2}x_{2,2} + \ldots + c_{2,n}x_{2,n} + \\ & \quad \ldots \\ & + c_{m,1}x_{m,1} + c_{m,2}x_{m,2} + \ldots + c_{m,n}x_{m,n}\end{aligned}$$

p.o.

$$\begin{aligned}& x_{1,1} + x_{1,2} + \ldots + x_{1,n} = 1 \\ & x_{2,1} + x_{2,2} + \ldots + x_{2,n} = 1 \\ & \ldots \\ & x_{m,1} + x_{m,2} + \ldots + x_{m,n} = 1\end{aligned}$$

$$\begin{aligned}& x_{1,1} + x_{2,1} + \ldots + x_{m,1} = 1 \\ & x_{1,2} + x_{2,2} + \ldots + x_{m,2} = 1 \\ & \ldots \\ & x_{1,n} + x_{2,n} + \ldots + x_{m,n} = 1\end{aligned}$$

$$x_{i,j} \in \{0, 1\},\ i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, n$$

Slično kao kod transportnog problema, i ovdje se javljaju *tri skupine ograničenja*: ograničenja na *izvršioce* (da svaki mora biti raspoređen na tačno jedno mjesto), ograničenja na *mjesta* (da na svako mjesto mora biti raspoređen tačno jedan izvršilac) i ograničenja na *logičku* (*binarnu*) *prirodu svih promjenljivih*. Koristeći oznake za sumaciju, ovaj matematski model se može kompaktnije zapisati na sljedeći način:

$$\arg\min Z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{i,j}\ x_{i,j}$$

p.o.

$$\sum_{j=1}^{n} x_{i,j} = 1\,,\ \ i = 1 \,..\, m$$

$$\sum_{i=1}^{m} x_{i,j} = 1\,,\ \ j = 1 \,..\, n$$

$$x_{i,j} \in \{0, 1\},\ \ i = 1 \,..\, m,\ \ j = 1 \,..\, n$$

Vidimo da po formi ovaj problem *jako podsjeća na transportni problem* kod kojeg su kapaciteti svih skladišta i potrošača *jednaki jedinici*, odnosno kod kojeg su *ponude* iz svih ishodišta *jednake jedinici*, odnosno sve *potrebe* odredišta su *jednake jedinici*. Međutim, razlika se javlja u ograničenjima na prirodu

promjenljivih $x_{i,j} \in \{0, 1\}$, $i = 1 .. m$, $j = 1 .. n$ zbog kojih problem, u obliku kakvom je gore postavljen, *uopće nije problem linearnog programiranja*. Zaista, ograničenja oblika $x_{i,j} \in \{0, 1\}$ se mogu zapisati u obliku $x_{i,j}(x_{i,j} - 1) = 0$ odnosno $x_{i,j}^2 = x_{i,j}$, iz čega direktno slijedi da su ova ograničenja u suštini *kvadratna* a ne linearna. Štaviše, ovo uopće nije ni problem *konveksnog programiranja*, s obzirom da dopustiva oblast *nije konveksan skup* (već se sastoji samo od diskretnih tačaka). Inače, problemi u kojima su sve promjenljive ograničene samo na vrijednosti 0 ili 1 često se nazivaju i ***logičko*** ili **0–1** ***programiranje***.

Ograničenja $x_{i,j} \in \{0, 1\}$ se mogu zamijeniti skupinom ograničenja $x_{i,j} \geq 0$, $x_{i,j} \leq 1$ i $x_{i,j} \in \mathbf{Z}$, od kojih su prva dva linearna, tako da ostaje samo još dodatno ograničenje na *cjelobronost* promjenljivih. Slijedi da problem raspoređivanja spada u porodicu problema *cjelobrojnog linearnog programiranja*, tačnije u podklasu ovih problema poznatu kao **0–1** ***linearno programiranje***.

Ne treba da nas zavara kombinatorna priroda ograničenja da sve promjenljive mogu biti samo 0 ili 1 pa da pomislimo da se problem može riješiti ispitivanjem *svih kombinacija vrijednosti* koje mogu uzeti ove promjenljive, zbog činjenice da broj tih kombinacija može biti nevjerovatno velik. Naime, kako problem raspoređivanja pri $m = n$ ima n^2 promjenljivih, broj takvih kombinacija iznosi 2^{n^2}, što je *mnogo gore* od $n!$ kombinacija koliko smo ih imali pri klasičnom kombinatornom pristupu rješavanju problema. To čini takav pristup *beznadežim* već pri posve malim vrijednostima n. Na primjer, već za $n = 10$ broj kombinacija koje bi trebalo ispitati iznosi $2^{100} = 1267650600228229401496703205376$. Bez obzira na to, koristeći neke *specijalne tehnike* za rješavanje problema *cjelobrojnog programiranja* kao što su recimo ***metod grananja i odsjecanja*** (engl. *branch-and-bound method*), koji ćemo obrađivati kasnije kada budemo razmatrali *opće probleme cjelobrojnog programiranja*, broj kombinacija bi se mogao svesti na razumnu mjeru koja bi se mogla obraditi u *razumnom vremenu*. Naime, ideja ovog metoda je zasnovana na ispitivanju samo onih kombinacija za koje se može na osnovu nekog kriterija *garantirati* da su *bolje* od ostalih kombinacija, odnosno na *odbacivanju neperspektivnih kombinacija*. Međuti,, uskoro ćemo vidjeti da za rješavanje problema raspoređivanja postoje *znatno pogodniji i efikasniji postupci*. Zapravo, prvo ćemo pokazati da su problemi raspoređivanja, bez obzira na ograničenja tipa $x_{i,j} \in \{0, 1\}$, ipak *specijalni slučajevi* transportnih problema (samim tim i problema linearnog programiranja), ali ćemo isto tako pokazati da ni njihovo rješavanje metodima za rješavanje transportnih problema (a pogotovo simpleks metodom) nije dobra ideja.

Sve do sada, razmatrali smo slučaj kada su broj izvršilaca i broj mjesta *jednaki*, odnosno kada je $m = n$. Takvi problemi raspoređivanja nazivaju se ***balansirani*** odnosno ***zatvoreni problemi raspoređivanja***. Na sličan način kao kod transportnog problema lako je pokazati da je $m = n$ *neophodan uvjet* da bi problem, u obliku kako je gore predstavljen, *uopće imao rješenja* (vidjećemo da je ovaj uvjet ne samo neophodan nego i *dovoljan*). Zaista, slično kao kod transportnog problema, saberemo li sva ograničenja na izvršioce kao i sva ograničenja na mjesta, dobićemo dva zbirna ograničenja čije su lijeve strane jednake, dok su im desne strane jednake m (kao suma m jedinica) odnosno n (kao suma n jedinica respektivno), odakle slijedi da mora biti $m = n$.

Ukoliko broj izvršilaca i broj mjesta *nisu jednaki*, govorimo o ***nebalansiranom*** odnosno ***otvorenom problemu raspoređivanja***. Kod ovakvih problema, matematski model se mora neznatno izmijeniti, jer smo vidjeli da gore navedeni model *uopće nije rješiv* ako je $m \neq n$. Razmotrimo prvo slučaj u kojem je *broj izvršilaca veći od broja mjesta*, odnosno u kojem je $m > n$. U tom slučaju, na svako mjesto može i dalje biti raspoređen jedan izvršilac, ali će postojati izvršiocia koji *neće biti raspoređeni nigdje*. To znači da u ograničenjima za ma kojeg j-tog izvršioca suma promjenljivih $x_{i,j}$, $i = 1, 2, ..., m$ ne mora biti isključivo 1, nego *može biti i nula* (ukoliko taj izvšilac nije raspoređen nigdje). Slijedi da se ograničenja za izvršioce mijenjaju i dobijaju oblik

$$\sum_{j=1}^{n} x_{i,j} \in \{0, 1\}, \quad i = 1 .. m$$

Ovim ograničenjima može se dati jednostavniji oblik. Prvo, treba primijetiti da zbog činjenice da sve promjenljive mogu biti samo 0 ili 1, suma sa lijeve strane svakako nikada ne može biti negativna, niti može biti broj koji nije cijeli broj. Slijedi da su, u prisustvu ograničenja da sve promjenljive mogu imati samo vrijednosti 0 ili 1, prethodna ograničenja ekvivalentna ograničenjima

$$\sum_{j=1}^{n} x_{i,j} \leq 1, \quad i = 1 .. m$$

Zaista, ako suma mora biti manja ili jednaka 1, kako nalažu ova ograničenja, a ne može biti ni manja od nule, niti neki broj koji nije cijeli, slijedi da ona može biti samo 0 ili 1, što su zahtijevala i prethodna ograničenja. Slijedi da se matematski model otvorenog problema raspoređivanja kod kojeg je broj izvršilaca veći od broja mjesta može predstaviti u sljedećem obliku:

$$\arg\min Z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{i,j}\ x_{i,j}$$

p.o.

$$\sum_{j=1}^{n} x_{i,j} \le 1\,,\ \ i = 1\,..\,m$$

$$\sum_{i=1}^{m} x_{i,j} = 1\,,\ \ j = 1\,..\,n$$

$$x_{i,j} \in \{0,1\},\ \ i = 1\,..\,m,\ \ j = 1\,..\,n$$

Pokažimo sada kako ovaj problem možemo svesti na *balansirani problem raspoređivanja*. Pokušamo li primijeniti isti pristup kao kod transportnih problema koji se zasniva na uvođenju m izravnavajućih promjenljivih $x_{i,\,n+1}$, $i = 1\,..\,m$, a koje će odgovarati ***fiktivnom mjestu*** (na koje će biti raspoređeni izvršioci koji zapravo *neće biti nigdje raspoređeni*) to će nas na kraju dovesti do dodatnog ograničenja

$$\sum_{i=1}^{m} x_{i,n+1} = m - n$$

koje se *ne uklapa* u opći model balansiranog problema raspoređivanja, jer koeficijent sa desne strane u općem slučaju *nije jednak jedinici* (osim ako je $m = n + 1$). Ovo je uostalom i logično, jer ukoliko je $m > n$, tada će $m - n$ izvršilaca ostati neraspoređeno, odnosno na fiktivno mjesto će biti raspoređeno $m - n$ izvršilaca. Da bismo postigli da problem bude *ekvivalentan balansiranom problemu raspoređivanja*, umjesto jednog fiktivnog mjesta uvešćemo $m - n$ *različih fiktivnih mjesta*, tako da će svaki od izvršilaca koji ne može biti raspoređen biti "raspoređen" na različito fiktivno mjesto (tako da će na svako fiktivno mjesto biti raspoređen tačno jedan izvršilac). U matematičkom modelu, ovo odgovara uvođenju $m\,(m - n)$ fiktivnih promjenljivih $x_{i,\,j}$, $i = 1\,..\,m$, $j = n + 1\,..\,m$. Naravno, efekat raspoređivanja na fiktivno mjesto jednak je nuli, odnosno imamo $c_{i,\,j} = 0$ za $i = 1\,..\,m$ i $j = n + 1\,..\,m$. Na ovaj način, problem je sveden na balansirani problem raspoređivanja.

Na sličan način se mogu modelirati nebalansirani problemi raspoređivanja kod kojih je *broj mjesta veći od broja izvršilaca*, tj. kod kojeg je $n > m$. Jasno je da će u takvim slučajevima postojati *mjesta na koje neće biti raspoređen niko*, tj. *nijedan izvršilac*. Sličnim rezonom kao u prethodnom slučaju, zaključujemo da se u takvim problemima u ograničenjima za mjesta javljaju nejednakosti tipa "manje ili jednako" umjesto jednakosti. Ovi problemi se svode na balansirane uvođenjem $n - m$ ***fiktivnih izvršilaca*** koji će biti "raspoređeni" na ona mjesta na koja neće biti zapravo niko raspoređen. Efekat ovakvih raspoređivanja je, naravno, i u ovom slučaju jednak nuli. U nastavku ćemo, bez umanjenja općenitosti, razmatrati *isključivo balansirane probleme raspoređivanja* (odnosno, pretpostavljaćemo da je $m = n$) s obzirom da smo upravo pokazali da se nebalansirani problemi raspoređivanja bez umanjenja općenitosti *uvijek mogu svesti na ekvivalentne balansirane probleme*.

U slučajevima da je iz nekog razloga *zabranjeno* nekog specifičnog izvršioca I_p rasporediti na neko specifično mjesto M_q, slično kao i kod transportnih problema možemo uvesti *kazneni koeficijent* koji će spriječiti da se ostvari takav raspored. Drugim riječima, za probleme minimizacije stavićemo $c_{p,q} = M$, a za probleme maksimizacije stavićemo $c_{p,q} = -M$, gdje je M neki *vrlo veliki broj*.

Svođenje problema raspoređivanja na transportni problem

Vidjeli smo da matematski zapis problema raspoređivanja *upadljivo podsjeća* na matematski zapis transportnog problema kod kojeg su kapaciteti svih skladišta i zahtjevi potrošača *jednaki jedinici*. Jedina razlika je što se u transportnom problemu javljaju ograničenja tipa $x_{i,\,j} \ge 0$, dok su u problemu raspoređivanja ova ograničenja zamijenjena znatno restriktivnijim ograničenjima tipa $x_{i,\,j} \in \{0,1\}$. Razmotrimo sada transportni problem koji dobijamo tako što u problemu raspoređivanja prosto *zamijenimo* ograničenja tipa $x_{i,\,j} \in \{0,1\}$ sa ograničenjima tipa $x_{i,\,j} \ge 0$, odnosno transportni problem oblika

$$\arg\min Z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j}\, x_{i,j}$$

p.o.

$$\sum_{j=1}^{n} x_{i,j} = 1\,,\ \ i = 1\,..\,n$$

$$\sum_{i=1}^{n} x_{i,j} = 1\,,\ \ j = 1\,..\,n$$

$$x_{i,j} \geq 0,\ \ i = 1\,..\,n,\ \ j = 1\,..\,n$$

Pokazaćemo da je, u izvjesnom smislu, ovaj transportni problem *ekvivalentnan* razmatranom problemu raspoređivanja. Tačnije, ukoliko ovaj trasnportni problem ima *jedinstveno optimalno rješenje*, ono je ujedno i jedinstveno optimalno rješenje razmatranog problema raspoređivanja, i obrnuto. Ukoliko ovaj transportni problem ima više različitih optimalnih rješenja, tada su optimalna rješenja razmatranog problema raspoređivanja *sva bazna optimalna rješenja ovog transportnog problema*, odnosno ona optimalna rješenja koja se nalaze *u vršnim tačkama* (s obzirom da u slučaju kada optimalno rješenje nije jedinstveno, mogu postojati i optimalna rješenja koja *nisu bazna*, odnosno koja se *ne nalaze* u vršnim tačkama).

Da bismo ovo pokazali, prvo treba uočiti da sva dopustiva rješenja ovog transportnog problema moraju zadovoljavati ne samo ograničenja $x_{i,j} \geq 0$, $i = 1\,..\,m$, $j = 1\,..\,n$ nego i ograničenja $x_{i,j} \leq 1$, $i = 1\,..\,m$, $j = 1\,..\,n$. Zaista, kada bi ma koja promjenljiva imala vrijednost veću od 1, zbirovi koji se javljaju u ograničenjima ovog transportnog problema ne bi mogli biti jednaki jedinici osim ako bi neka od promjenljivih bila *negativna*, što nije moguće zbog uvjeta $x_{i,j} \geq 0$. Slijedi da *sva dopustiva rješenja* ovog transportnog problema zadovoljavaju $0 \leq x_{i,j} \leq 1$, $i = 1\,..\,m$, $j = 1\,..\,n$. Naravno, iz ovoga *ne slijedi* da u svim dopustivim rješenjima ovog transportnog problema promjenljive $x_{i,j}$ moraju imati vrijednosti isključivo 0 ili 1, Recimo, rješenje $x_{i,j} = 1/n$, $i = 1\,..\,m$, $j = 1\,..\,n$ je očigledno dopustivo, a u njemu promjenljive imaju vrijednosti različite od 0 i 1. Međutim, ranije smo pokazali da su sva *bazna* dopustiva rješenja transportnih problema cjelobrojna ukoliko su svi kapaciteti cijeli brojevi. Kako su u ovom transportnom problemu svi kapaciteti cijeli brojevi (tačnije, svi su *jedinice*), slijedi da su sva njegova bazna rješenja također cjelobrojna. Međutim, kako su jedini cijeli brojevi $x_{i,j}$ koji zadovoljavaju uvjet $0 \leq x_{i,j} \leq 1$ zapravo 0 ili 1, slijedi da sva *bazna* dopustiva rješenja ovog transportnog problema zadovoljavaju i uvjet $x_{i,j} \in \{0,1\}$, odnosno *zadovoljavaju i polazni problem raspoređivanja*. Posljedica toga je da se *optimalna rješenja* polaznog problema raspoređivanja poklapaju sa *baznim optimalnim rješenjima* ovog transportnog problema. Specijalno, ukoliko ovaj transportni problem ima jedinstveno optimalno rješenje, ono je ujedno i jedinstveno rješenje polaznog problema raspoređivanja.

Primijetimo ipak da dobijeni transportni problem *nije u potpunosti ekvivalentan* polaznom problemu raspoređivanja. Zaista, ukoliko optimalno rješenje dobijenog transportnog problema nije jedinstveno, ovaj problem može imati i *necjelobrojna optimalna rješenja* koja naravno ne zadovoljavaju uvjet $x_{i,j} \in \{0,1\}$. Međutim, ukoliko dobijeni transportni problem rješavamo bilo kojim metodom koji pretražuje *isključivo bazna rješenja* (takvi su svi metodi koji su u osnovi *zasnovani na simpleks metodu*, uključujući i *sve metode koje smo razmatrali*), nađena rješenja dobijenog transportnog problema biće cjelobrojna, pa će samim tim zadovoljavati i polazni problem raspoređivanja. Slijedi da se, makar principijelno, problemi raspoređivanja mogu rješavati primjenom metoda za rješavanje transportnih problema koji pretražuju isključivo bazna rješenja na dobijeni (polu)ekvivalentni transportni problem. Za tu svrhu se *ne smiju* koristiti metodi koji mogu kao rezultat dati optimalno rješenje koje *nije bazno*, kao što su razni *metodi unutrašnje tačke* (takvi metodi mogu dati necjelobrojno rješenje koje zadovoljava dobijeni transportni problem, ali ne i polazni problem raspoređivanja).

Bez obzira na činjenicu da se problemi raspoređivanja mogu svesti na transportne probleme, njihovo rješavanje rješavanjem ekvivalentnog transportnog problema je *vrlo loša strategija*. Da bismo uvidjeli razlog za to, primijetimo da *svako dozvoljeno rješenje* balansiranog problema raspoređivanja formata $n \times n$ uvijek ima *tačno n promjenljivih različitih od nule* (i koje su tada jednake jedinici), dok bazna rješenja općih balansiranih transportnih problema formata $n \times n$ mogu imati do $2n-1$ nenultih promjenljivih. Slijedi da je svako dozvoljeno rješenje problema raspoređivanja *degenerirano bazno rješenje* ekvivalentnog transportnog problema sa *visokim stepenom degeneracije* $n-1$. Drugim riječima, transportni problem koji se dobija iz polaznog problema raspoređivanja je takav da je *svako njegovo bazno rješenje vrlo visoko degenerirano*. Posljedica je da će primjena svakog od metoda koje su u osnovi zasnovani na simpleks metodu biti praćena

čestim *zastojima* gdje će se dešavati da se izvrši čitav niz uzastopnih iteracija u kojima se funkcija cilja *neće poboljšavati*, nego ćemo samo prelaziti iz jednog u drugo bazno rješenje, koje odgovara jednoj te istoj *degeneriranoj vršnoj tački* (kod ranije opisanih namjenskih metoda za rješavanje transportnih problema, ovo se svodi na prebacivanje *fiktivnih infinitezimalnih transporta* sa jednog na drugo mjesto, bez ikakvih stvarnih promjena).

Detaljnije analize pokazuju da kod ekvivalentnih transportnih problema na koji se svode problemi raspoređivanja broj različitih degeneriranih baznih rješenja koje odgovaraju istoj degeneriranoj vršnoj tački može biti *eksponencijalna funkcija* od *n*, tako da, u najgorem slučaju, metodi zasnovani na simpleks metodu mogu potrošiti *eksponencijalno mnogo iteracija* da bi iz jedne degenerirane vršne tačke prešli u drugu vršnu tačku koja se razlikuje od nje (ali koja će *ponovo biti degenerirana*). Posljedica ovoga je da svi metodi koji su u osnovi zasnovani na simpleks metodu često imaju vrijeme izvršavanja koje je *eksponencijalna funkcija od n* kada se primijene na ekvivalentni transportni problem koji se dobija iz polaznog problema raspoređivanja, što je neprihvatljivo. Zbog toga su za rješavanje problema raspoređivanja *razvijeni posebni metodi*, koje ćemo nešto kasnije obraditi.

Mada se problemi raspoređivanja u praksi praktično nikad ne rješavaju svođenjem na ekvivalentni transportni problem, u sljedećem primjeru ćemo postupiti *upravo tako*, i to iz tri razloga. Prvi razlog je da demonstriramo *pojavu zastoja* o kojoj smo govorili. Drugi razlog je da demonstriramo *kako uopće teče postupak rješavanja transportnih problema u slučaju pojave višestruke degeneracije*. Konačno, treći razlog je što će se prilikom rješavanja pokazati da *nisu tačna neka uobičajena vjerovanja* za koja se ponekad u literaturi navodi da vrijede za simpleks metod i sve druge metode zasnovane na njemu.

- ***Primjer***: Tri izvršioca I_1, I_2 i I_3 treba rasporediti na četiri radna mjesta M_1, M_2, M_3 i M_4, pri čemu su dobiti koje se postižu raspoređivanjem svakog od kandidata na svako od raspoloživih radnih mjesta date u sljedećoj tabeli:

	M_1	M_2	M_3	M_4
I_1	3	2	5	4
I_2	6	4	7	8
I_3	1	6	3	7

  Potrebno je odrediti otpimalan raspored ovih izvršilaca na radna mjesta kojim će se ostvariti *maksimalna ukupna dobit*. Rješavanje izvesti svođenjem na ekvivalentni transportni problem i primjenom *MODI* metoda na početno rješenje dobijeno metodom sjeverozapadnog ugla.

U ovom primjeru, očigledno se radi o *nebalansiranom problemu raspoređivanja*, jer imamo jednog izvršioca manje u odnosu na broj raspoloživih radnih mjesta. Zbog toga ćemo problem *balansirati* uvođenjem *fiktivnog izvršioca* I_4 (koji ne proizvodi *nikakvu dobit* gdje god da se rasporedi). Dalje ćemo ovaj problem posmatrati kao problem transporta opisan sljedećom tabelom:

	M_1	M_2	M_3	M_4	Zalihe
I_1	3	2	5	4	1
I_2	6	4	7	8	1
I_3	1	6	3	7	1
I_4	0	0	0	0	1
Potrebe	1	1	1	1	

Ne treba zaboraviti da se radi o *problemu maksimizacije*. Stoga bi sve koeficijente *trebalo pomnožiti sa* –1 da ga *svedemo na problem minimizacije*. Međutim, moguće je transportne probleme rješavati i *bez ove transformacije*, samo je potrebno pri odluci na koje polje preraspodijeliti tranport tražiti *najpozitivniji* a ne najnegativniji relativni koeficijent troškova (ove koeficijente bi u ovom slučaju bilo bolje zvati ***relativni koeficijenti dobiti***). Tako ćemo i ovdje postupiti.

Nađimo prvo *dozvoljeno bazno rješenje* prema metodu sjeverozapadnog ugla. Nije na odmet napomenuti da bi u ovom primjeru *metod maksimalne jedinične dobiti* (ovo je pandan metoda minimalnih jediničnih troškova za probleme maksimizacije, u kojem se umjesto polja koje ima minimalnu vrijednost daje prioritet polju koje ima maksimalnu vrijednost) odnosno *Vogelov metod* (na prikadan način adaptiran za probleme maksimizacije, tako da se gledaju najveći a ne najmanji elementi u onim redovima i/ili kolonama koje imaju maksimalno žaljenje) slučajno *odmah dali optimalno rješenje*, tako da dalje ne bismo imali mnogo

posla. Međutim, upravo da bismo demonstrirali kako teče postupak *unapređivanja rješenja* koje nije optimalno, postupićemo kako je traženo u postavci, odnosno početno rješenje ćemo naći metodom sjeverozapadnog ugla.

Pođimo od polja $x_{1,1}$. Ovdje imamo $x_{1,1} = \min\{1,1\} = 1$. Ovim se istovremeno potpuno iscrpljuje "skladište" I_1 i podmiruje "potrošač" M_1, što odmah znači da će se dobiti *degenerirano početno rješenje*. Da ne bismo kasnije tražili gdje uvoditi fiktivne infinitezimalne transporte, odlučimo se (nasumice) da u potpunosti podmirimo "potrošača" M_1, a da u "skladištu" I_1 *ostavimo infinitezimalnu količinu* "robe" ε. Prelazimo sada na polje $x_{1,2}$, za koje imamo $x_{1,2} = \min\{\varepsilon, 1\} = \varepsilon$. Ovo sada potpuno iscrpljuje "skladište" I_1, dok potražnja "potrošača" M_2 ostaje praktično ista, jer je $1 - \varepsilon = 1$. Dalje nastavljamo istim postupkom. Imamo $x_{2,2} = \min\{1,1\} = 1$, čime se ponovo potpuno iscrpljuje "skladište" I_2 i podmiruje "potrošač" M_2. Opet ćemo se odlučiti da u potpunosti podmirimo "potrošača" M_2, a ostavićemo infinitezimalnu količinu "robe" ε u "skladištu" I_2. Nakon ovoga imamo $x_{2,3} = \min\{\varepsilon, 1\} = \varepsilon$ što potpuno iscrpljuje "skladište" I_2 a praktično ne mijenja potrebe "potrošača" M_3. U sljedećem koraku je $x_{3,3} = \min\{1, 1\} = 1$. Podmirićemo "potrošača" M_3 i ostaviti infinitezimalnu količinu robe u "skladištu" I_3. Analogno, dalje ćemo imati $x_{3,4} = \min\{\varepsilon, 1\} = \varepsilon$ i, konačno, $x_{4,4} = \min\{1, 1\} = 1$. Ovim se dobija dozvoljeno bazno rješenje prikazano u sljedećoj tabeli, kojem odgovara cijena "transporta" $Z = 3 + 4 + 1 + 1 = 10$ (s obzirom da se transport sa iznosom ε tretira kao nulti trasnport):

	M_1	M_2	M_3	M_4
I_1	1 / 3	ε / 2	5	4
I_2	6	1 / 4	ε / 7	8
I_3	1	6	1 / 3	ε / 7
I_4	0	0	0	1 / 0

U ovoj tabeli već imamo onoliko infinitezimalnih transporta koliko je potrebno da bismo mogli dalje nastaviti sa *MODI* metodom. Nasumice ćemo se odlučiti da stavimo $u_1 = 0$, nakon čega imamo $v_1 = 3 - u_1 = 3$, $v_2 = 2 - u_1 = 2$, $u_2 = 4 - v_2 = 2$, $v_3 = 7 - u_2 = 5$, $u_3 = 3 - v_3 = -2$, $v_4 = 7 - u_3 = 9$ i $u_4 = 0 - v_4 = -9$. Ovo je najlakše uraditi direktno na tabeli:

	M_1	M_2	M_3	M_4	u_i
I_1	1 / 3	ε / 2	5	4	0
I_2	6	1 / 4	ε / 7	8	2
I_3	1	6	1 / 3	ε / 7	–2
I_4	0	0	0	1 / 0	–9
v_j	3	2	5	9	

Sada trebamo na dobro poznati način odrediti relativne koeficijente dobiti. Dobijamo $d_{1,3} = 0$, $d_{1,4} = -5$, $d_{2,1} = 1$, $d_{2,4} = -3$, $d_{3,1} = 0$, $d_{3,2} = 6$, $d_{4,1} = 6$, $d_{4,2} = 7$ i $d_{4,3} = 4$. Najpozitivniji (najveći) koeficijent relativne dobiti je $d_{4,2}$, tako da je potrebno naći ciklus koji počinje od polja $x_{4,2}$ preko polja sa nenultim transportom. Takav ciklus glasi $x_{4,2} \to x_{4,4} \to x_{3,4} \to x_{3,3} \to x_{2,3} \to x_{2,2} \to x_{4,2}$. Prilikom preraspodjele transporta na polje $x_{4,2}$ umanjivaće se transporti $x_{4,4}$, $x_{3,3}$ i $x_{2,2}$, tako da je $t_{max} = \min\{1, 1, 1\} = 1$. Nakon preraspodjele t_{max} količinskih jedinica imaćemo $x_{4,2} = x_{3,4} = x_{2,3} = 1$ (jer je $\varepsilon + 1 = 1$), dok će sva tri transporta $x_{4,4}$, $x_{3,3}$ i $x_{2,2}$ pasti na nulu. To znači da će, ukoliko anuliramo sva ta tri transporta, u novom rješenju *ponovo nedostajati dva polja sa nenultim transportom* da bismo mogli nastaviti dalje. Da bismo izbjegli naknadno traženje gdje ubaciti fiktivne infinitezimalne transporte bez kojih nećemo moći nastaviti dalje, odlučimo se da *samo jedan od ova tri transporta zaista anuliramo,* recimo transport $x_{4,4}$, dok ćemo na poljima $x_{3,3}$ i $x_{2,2}$ ostaviti infinitezimalne transporte. Time dobijamo novo rješenje prikazano u sljedećoj tabeli, u kojoj smo, radi nastavka postupka, odmah izračunali i odgovarajuće vrijednosti dualnih promjenljivih:

	M_1	M_2	M_3	M_4	u_i
I_1	1 / 3	ε / 2	5	4	2
I_2	6	ε / 4	1 / 7	8	4
I_3	1	6	ε / 3	1 / 7	0
I_4	0	1 / 0	0	0	0
v_j	1	0	3	7	

Novom baznom rješenju odgovara vrijednost funkcije cilja $Z = 3 + 7 + 7 + 0 = 17$, odakle vidimo da se funkcija cilja *poboljšala*. Sada iz ove tabele dobijamo $d_{1,3} = 0$, $d_{1,4} = –5$, $d_{2,1} = 1$, $d_{2,4} = –3$, $d_{3,1} = 0$, $d_{3,2} = 6$, $d_{4,1} = –1$, $d_{4,3} = –3$ i $d_{4,4} = –7$. Vidimo da je najveći relativni koeficijent dobiti $d_{3,2}$. Ciklus od polja $x_{3,2}$ glasi $x_{3,2} \to x_{3,3} \to x_{2,3} \to x_{2,2} \to x_{3,2}$. Preraspodjelom transporta na polje $x_{3,2}$ umanjuju se trasnporti $x_{3,3}$ i $x_{2,2}$, te je $t_{max} = \min\{\varepsilon, \varepsilon\} = \varepsilon$. Slijedi da ćemo ovaj put preraspodijeliti samo *infinitezimalnu količinu transporta* (efektivno, zapravo nećemo uraditi *ništa*). Nakon preraspodjele ε količinskih jedinica (tj. ničega) imaćemo $x_{3,2} = \varepsilon$ i $x_{2,3} = 1 + \varepsilon = 1$, dok se $x_{3,3}$ i $x_{2,2}$ umanjuju za ε, i kako su oba bila jednaka ε, oba će se anulirati (zbog čega bi nam za nastavak nedostajalo jedno polje sa nenultim transportom). Odlučimo se da samo $x_{3,3}$ stvarno anuliramo, dok ćemo $x_{2,2}$ ostaviti na infinitezimalnoj vrijednosti. Time dobijamo sljedeću tabelu, koja se razlikuje od prethodne *samo po tome kako su razmješteni infinitezimalni transporti*:

	M_1	M_2	M_3	M_4	u_i
I_1	1 / 3	ε / 2	5	4	2
I_2	6	ε / 4	1 / 7	8	4
I_3	1	ε / 6	3	1 / 7	6
I_4	0	1 / 0	0	0	0
v_j	1	0	3	1	

Vrijednost funkcije cilja je naravno *ostala ista*, jer se *ništa suštinski nije promijenilo*. Sada imamo $d_{1,3} = 0$, $d_{1,4} = 1$, $d_{2,1} = 1$, $d_{2,4} = 3$, $d_{3,1} = –6$, $d_{3,3} = –6$, $d_{4,1} = –1$, $d_{4,3} = –3$ i $d_{4,4} = –1$. Najveći relativni koeficijent dobiti je $d_{2,4}$. Odgovarajući ciklus od polja $x_{2,4}$ glasi $x_{2,4} \to x_{2,2} \to x_{3,2} \to x_{3,4} \to x_{2,4}$, tako da imamo $t_{max} = \min\{\varepsilon, 1\} = \varepsilon$. Ovo znači da ćemo ponovo preraspodijeliti infinitezimalnu količinu transporta. Zbog činjenice da je $\varepsilon + \varepsilon = \varepsilon$, $1 + \varepsilon = 1$ i $1 - \varepsilon = 1$, jedino što će se nakon preraspodjele desiti je što će postati $x_{2,4} = \varepsilon$, dok će se $x_{2,2}$ anulirati. Ovim dolazimo do sljedeće tabele:

	M_1	M_2	M_3	M_4	u_i
I_1	1 / 3	ε / 2	5	4	2
I_2	6	4	1 / 7	ε / 8	7
I_3	1	ε / 6	3	1 / 7	6
I_4	0	1 / 0	0	0	0
v_j	1	0	0	1	

Funkcija cilja i dalje ostaje ista. Dalje imamo $d_{1,3} = 3$, $d_{1,4} = 1$, $d_{2,1} = –2$, $d_{2,2} = –3$, $d_{3,1} = –6$, $d_{3,3} = –3$, $d_{4,1} = –1$, $d_{4,3} = 0$ i $d_{4,4} = –1$. Najveći relativni koeficijent dobiti je $d_{1,3}$, dok odgovarajući ciklus od polja $x_{1,3}$ glasi $x_{1,3} \to x_{1,2} \to x_{3,2} \to x_{3,4} \to x_{2,4} \to x_{2,3} \to x_{1,3}$. Dalje je $t_{max} = \min\{\varepsilon, 1, 1\} = \varepsilon$. Nakon što se obavi preraspodjela, jedino što se mijenja je što će postati $x_{1,3} = \varepsilon$, dok će se $x_{1,2}$ anulirati. Time dolazimo do situacije prikazane u sljedećoj tabeli:

	M_1	M_2	M_3	M_4	u_i
I_1	1 / 3	2	ε / 5	4	0
I_2	6	4	1 / 7	ε / 8	2
I_3	1	ε / 6	3	1 / 7	1
I_4	0	1 / 0	0	0	–5
v_j	3	5	5	6	

Ni ovaj put nismo napravili nikakav progres po pitanju unapređenja funkcije cilja, odnosno potrošili smo već tri iteracije samo *premještajući infinitezimalne transporte*. Ovaj put ćemo imati više sreće. Iz ove tabele dobijamo $d_{1,2} = -3$, $d_{1,4} = -2$, $d_{2,1} = 1$, $d_{2,2} = -3$, $d_{3,1} = -3$, $d_{3,3} = -3$, $d_{4,1} = 2$, $d_{4,3} = 0$ i $d_{4,4} = -1$. Sada imamo da je najveći koeficijent relativne dobiti $d_{4,1}$, dok odgovarajući ciklus koji počinje od polja $x_{4,1}$ glasi $x_{4,1} \to x_{4,2} \to x_{3,2} \to x_{3,4} \to x_{2,4} \to x_{2,3} \to x_{1,3} \to x_{1,1} \to x_{4,1}$. Dalje imamo je $t_{max} = \min\{1, 1, 1, 1\} = 1$, tako da konačno dobijamo priliku da izvršimo *neku konkretnu preraspodjelu*. Nakon preraspodjele ove količine, imaćemo $x_{4,1} = x_{3,2} = x_{2,4} = x_{1,3} = 1$, dok se sva četiri transporta $x_{4,2}$, $x_{3,4}$ i $x_{2,3}$ i $x_{1,1}$ anuliraju. Ponovo ćemo se, da bismo mogli nastaviti dalje bez potrebe da tragamo gdje staviti infinitezimalne transporte, odlučiti da potpuno anuliramo *samo jedan* od ova četiri transporta, recimo $x_{4,2}$, dok ćemo ostale transporte ostaviti na infinitezimalnom iznosu. Ovim dobijamo sljedeću tabelu:

	M_1	M_2	M_3	M_4	u_i
I_1	ε / 3	2	1 / 5	4	0
I_2	6	4	ε / 7	1 / 8	2
I_3	1	1 / 6	3	ε / 7	1
I_4	1 / 0	0	0	0	–3
v_j	3	5	5	6	

Funkcija cilja se sada poboljšala i iznosi $Z = 5 + 8 + 6 + 0 = 19$. Zapravo, vidjećemo uskoro da je rješenje koje smo upravo dobili *optimalno*, ali je interesantno da algoritam to *još neće primijetiti* (kasnije ćemo objasniti šta je razlog za ovakvo prividno neočekivano ponašanje). Zaista, iz ove tabele slijede relativni koeficijenti dobiti $d_{1,2} = -3$, $d_{1,4} = -2$, $d_{2,1} = 1$, $d_{2,2} = -3$, $d_{3,1} = -3$, $d_{3,3} = -3$, $d_{4,2} = -2$, $d_{4,3} = -2$ i $d_{4,4} = -3$. Kako među ovim koeficijentima još uvijek ima pozitivnih, algoritam se *još uvijek ne završava*. Najveći koeficijent je $d_{2,1}$, tako da treba potražiti ciklus koji počinje od polja $x_{2,1}$. Takav ciklus je $x_{2,1} \to x_{2,3} \to x_{1,3} \to x_{1,1} \to x_{2,1}$. Međutim, njemu odgovara maksimalna dozvoljena preraspodjela transporta $t_{max} = \min\{\varepsilon, \varepsilon\} = \varepsilon$, koja je ponovo "presipanje iz šupljeg u prazno". Ova preraspodjela dovodi do toga da će postati $x_{2,1} = \varepsilon$, dok se $x_{2,3}$ i $x_{1,1}$ anuliraju. Odlučimo li se da anuliramo samo transport $x_{2,3}$ a $x_{1,1}$ ostavimo na infinitezimalnoj vrijednosti, dolazimo do sljedeće tabele:

	M_1	M_2	M_3	M_4	u_i
I_1	ε / 3	2	1 / 5	4	3
I_2	ε / 6	4	7	1 / 8	6
I_3	1	1 / 6	3	ε / 7	5
I_4	1 / 0	0	0	0	0
v_j	0	1	2	2	

Funkcija cilja se naravno nije promijenila, dok relativni koeficijenti dobiti sada iznose $d_{1,2} = -2$, $d_{1,4} = -1$, $d_{2,2} = -3$, $d_{2,3} = -1$, $d_{3,1} = -4$, $d_{3,3} = -4$, $d_{4,2} = -1$, $d_{4,3} = -2$ i $d_{4,4} = -2$. Konačno smo dobili da su svi relativni koeficijenti dobiti *negativni*, tako da algoritam *završava sa radom*. Dakle, optimalno rješenje glasi $x_{1,3} = x_{2,4} = x_{3,2} = x_{4,1} = 1$ i dok su sve ostale promjenljive $x_{i,j}$ za i i j u opsegu od 1 do 4 jednake nuli

(infinitezimalni transporti su također praktično jednaki nuli). Drugim riječima, izvršioca I_1 treba rasporediti na radno mjesto M_3, izvršioca I_2 na radno mjesto M_4, izvršioca I_3 na radno mjesto M_2, dok će radno mjesto M_1 *ostati neupražnjeno*, jer će na njega biti raspoređen *fiktivni izvršilac* I_4.

Sada se može postaviti pitanje *zbog čega algoritam nije u prethodnoj iteraciji primijetio da je dostignuto optimalno rješenje* nego je bila neophodna *još jedna iteracija*. Stvar je u rasprostranjenoj zabludi da *simpleks algoritam obavezno završava sa radom čim se dostigne optimalno rješenje*. Prava istina je da simpleks algoritam završava sa radom *tek kad rješenje postane dualno dopustivo*. Znamo da se dualna dopustivost uspostavlja tek pri dostizanju optimalnog rješenja. Međutim, u slučaju kada je optimalno rješenje *degenerirano*, njemu zapravo odgovara više praktično *identičnih* baznih rješenja, koji se razlikuju samo po tome koje su promjenljive sa vrijednosti 0 u bazi, a koje nisu (odnosno koji se transporti tretiraju kao infinitezimalni, a koji se tretiraju kao čista nula pri rješavanju transportnih problema). Pri tome, *ne mora značiti* da su sva ta bazna rješenja *dualno dopustiva* (iako makar jedno od njih mora biti). Desi li se da simpleks metod dođe do degeneriranog baznog rješenja koje slučajno *nije dualno dopustivo*, iteracije će se nastaviti sve dok se ne uspostavi dualna dopustivost, pri čemu se u tim iteracijama neće dešavati ništa pametno, osim razmjene promjenljivih u bazi koje su jednake nuli sa promjenljivim izvan baze, koje su svakako jednake nuli (što se pri rješavanju transportnih problema manifestira kao premještanje rasporeda infinitezimalnih transporta). To se upravo desilo prethodno razmotrenom primjeru. U svakom slučaju, ovaj primjer jasno ilustrira da probleme raspoređivanja nije ni u kom slučaju pametno rješavati općim metodama za rješavanje transportnih problema.

Cjelobrojni poliedri i totalna unimodularnost

Prije nego što razmotrimo specifične metode za rješavanje problema raspoređivanja, istražimo malo detaljnije *zbog čega transportni problemi sa cjelobrojnim kapacitetima uvijek imaju cjelobrojna optimalna rješenja*. Naime, upravo ova osobina nam je omogućila da pokažemo da se problemi raspoređivanja također mogu posmatrati kao specijalan slučaj problema linearnog programiranja. Nakon što steknemo bolji uvid u ovu problematiku, vidjećemo da se sličan pristup može iskoristiti da se pokaže i da se *neki drugi problemi također mogu posmatrati kao specijalan slučaj problema linearnog programiranja*, iako *na prvi pogled ne izgledaju tako*.

Prvo je potrebno da uvedemo neke pojmove. Za neki *n*-dimenzionalni poliedar u $\mathbb{R}^n$ kažemo da je ***cjelobrojni poliedar*** ukoliko mu svi vrhovi *imaju cjelobrojne koordinate*. Nas će uglavnom zanimati samo poliedri koji su *dopustive oblasti problema linearnog programiranja*. Tako su dopustive oblasti transportnih problema sa cjelobrojnim kapacitetima i transportnih problema koji se dobiju transformacijom problema raspoređivanja cjelobrojni. Cjelobrojni poliedri su interesantni zbog činjenice da ukoliko je dopustiva oblast nekog problema cjelobrojni poliedar, tada eventualno prisustvo zahtijeva na cjelobrojnost rješenja *ne mijenja način rješavanja problema*, odnosno eventualni zahtjevi na cjelobrojnost rješenja se u takvim problemima mogu prosto *izostaviti*.

Neka je dat poliedar $\Omega = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\,\mathbf{x} = \mathbf{b} \wedge \mathbf{x} \geq 0\}$ koji predstavlja dopustivu oblast nekog problema linearnog programiranja u kanonskom obliku. Pretpostavićemo da matrica $\mathbf{A}$ ima m redova i n kolona, pri čemu je $m \leq n$. Bez umanjenja općenitosti, smatraćemo da je matrica $\mathbf{A}$ *punog ranga*, tj. da ima rang m. Ukoliko to nije slučaj, neke jednačine iz sistema jednačina $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ se mogu *izbaciti* a da se rješenje sistema ne promijeni, nakon čega ćemo imati novu matricu koja će biti punog ranga. Dalje, ukoliko su elementi matrice $\mathbf{A}$ i vektora $\mathbf{b}$ *racionalni brojevi*, uvijek se jednačina $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ može pomnožiti *najmanjim zajedničkim sadržiocem* svih nazivnika u razlomcima koji figuriraju u $\mathbf{A}$ i $\mathbf{b}$ čime će svi elementi u $\mathbf{A}$ i $\mathbf{b}$ postati *cijeli brojevi*. Ukoliko među elementima koji figuriraju u $\mathbf{A}$ i $\mathbf{b}$ ima *iracionalnih brojeva*, ovo *nije uvijek moguće postići*, ali takvi slučajevi *nisu od interesa* za razmatranja koja slijede. Zbog toga ćemo bez umanjenja općenitosti smatrati da su svi elementi koji se javljaju u matrici $\mathbf{A}$ i vektoru $\mathbf{b}$ cijeli brojevi. Jasno je da ukoliko je Ω cjelobrojni poliedar, tada su *sva dopustiva rješenja odgovarajućeg problema linearnog programiranja cjelobrojna*.

Za matricu $\mathbf{A}$ čiji su *svi minori* jednaki 0, 1 ili –1, kažemo da je ***totalno unimodularna***. Totalna unimodularnost je veoma značajna zbog činjenice da ona predstavlja *dovoljan uvjet* da *svi poliedri oblika* $\Omega = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\,\mathbf{x} = \mathbf{b} \wedge \mathbf{x} \geq 0\}$ kod kojeg matrica $\mathbf{A}$ i vektor $\mathbf{b}$ imaju cjelobrojne koordinate *budu cjelobrojni, bez obzira na vektor* $\mathbf{b}$, odnosno ukoliko je matrica $\mathbf{A}$ totalno unimodularna, svi takvi poliedri Ω su *sigurno cjelobrojni*. Ovo je lako pokazati. Naime, ranije smo vidjeli da se svaka vršna tačka poliedra Ω može izraziti u obliku $\mathbf{x}_B = \mathbf{A}_B^{-1}\,\mathbf{b}$, $\mathbf{x}_N = 0$ gdje je matrica $\mathbf{A}_B$ formirana od onih kolona matrice $\mathbf{A}$ koje odgovaraju promjenljivim koje u toj vršnoj tački čine bazu. Naravno, ukoliko je matrica $\mathbf{A}_B$ singularna, to prosto znači da odgovarajuće promjenljive *ne mogu tvoriti bazu* (tada je $\det \mathbf{A}_B = 0$). Uzmimo sada da

matrica matrica $\mathbf{A}_B$ nije singularna. Sad, poznato je da je $\mathbf{A}_B^{-1} = \text{adj}\,\mathbf{A}_B / \det \mathbf{A}_B$. Kako je $\det \mathbf{A}_B$ zapravo neki od minora matrice **A**, a ona je po pretpostavci totalno unimodularna, to je $\det \mathbf{A}_B$ ili 1 ili –1 (zbog nesingularnosti, nula je ovdje isključena). Dalje, kako su elementi matrice $\mathbf{A}_B$ po pretpostavci cijeli brojevi, iz definicije adjungovane metrice matrica $\text{adj}\,\mathbf{A}_B$ također ima cjelobrojne koeficijente, tako da je i matrica $\mathbf{A}_B^{-1}$ cjelobrojna. Konačno, zbog pretpostavljene cjelobrojnosti vektora **b**, elementi vektora $\mathbf{x}_B = \mathbf{A}_B^{-1}\,\mathbf{b}$ će također biti cjelobrojni, odnosno vršna tačka će imati cjelobrojne koordinate što je i trebalo pokazati.

Treba napomenuti da je totalna unimodularnost *dovoljan* ali ne i *neophodan* uvjet za cjelobrojnost svih poliedara Ω gore navedenog oblika. Neophodan uvjet je da *svi minori reda m* budu jednaki 0, 1 ili –1, a *ne nužno svi minori*. Zaista, lako je vidjeti da prethodno rezonovanje vrijedi i ako su samo minori reda *m* jednaki 0, 1 ili –1, jer to garantira cjelobrojnost matrice $\mathbf{A}_B^{-1}$ kad god je matrica $\mathbf{A}_B$ nesingularna. S druge strane, ukoliko je makar jedan minor reda *m* različit od 0, 1 i –1, to znači da će makar za jednu bazu $\det \mathbf{A}_B$ biti različita od 1 i –1, te matrica $\mathbf{A}_B^{-1}$ *neće biti cjelobrojna*. Slijedi da se neće moći postići da za sve cjelobrojne vektore **b** vektor $\mathbf{x}_B = \mathbf{A}_B^{-1}\,\mathbf{b}$ bude cjelobrojan. Ipak, ukoliko matrica **A** sadrži *jediničnu submatricu* unutar sebe (što je uvijek slučaj kada se razmatraju problemi linearnog programiranja *u normalnom obliku*), tada se lako može pokazati da ukoliko su svi minori reda *m* jednaki 0, 1 ili –1, tada su *i svi ostali minori* istog oblika, odnosno **A** je tada i *totalno unimodularna*. Slijedi da je u takvim slučajevima totalna unimodularnost *potreban i dovoljan uvjet* za cjelobrojnost svih poliedara Ω gore navedenog oblika.

Uvjet totalne unimodularnosti *nije lagan za provjeravanje*. Međutim, u linearnoj algebri poznati su *brojni dovoljni uvjeti* koji garantiraju totalnu unimodularnost neke matrice. Jedan od poznatijih tvrdi da je matrica **A** sigurno totalno unimodularna ukoliko su zadovoljeni sljedeći uvjeti:

1. Svi elementi matrice **A** su *isključivo* 0, 1 *ili* –1;
2. Svaka kolona matrice ima *najviše dva elementa različita od nule*;
3. *Redovi matrice* se mogu *podijeliti u dva skupa* P i Q, tako da *nenulti elementi istog znaka iz ma koje kolone ne pripadaju istom skupu*, a *nenulti elementi različitog znaka iz ma koje kolone pripadaju istom skupu*.

Ovaj uvjet je dovoljan da se pokaže da *transportni problemi imaju totalno unimodularnu matricu*. Da bismo to uvidjeli, razmotrimo *kako uopće izgleda matrica* **A** kod transportnih problema. Ukoliko raspišemo sva ograničenja transportnog problema tako da istovjetne promjenljive budu potpisane *jedna ispod druge*, ta ograničenja izgledaju ovako:

$$
\begin{array}{llllll}
x_{1,1} + x_{1,2} + \ldots + x_{1,n} & & & & & = a_1 \\
& x_{2,1} + x_{2,2} + \ldots + x_{2,n} & & & & = a_2 \\
& & & \ldots & & \\
& & & & x_{m,1} + x_{m,2} + \ldots + x_{m,n} & = a_m \\
x_{1,1} & + x_{2,1} & & \ldots & + x_{m,1} & = b_1 \\
x_{1,2} & & + x_{2,2} & \ldots & + x_{m,2} & = b_2 \\
& & & \ldots & & \\
& x_{1,n} & + x_{2,n} & \ldots & + x_{m,n} & = b_n
\end{array}
$$

Odavde neposredno vidimo da matrica ograničenja kod transportnih problema ima sljedeći oblik:

$$
\mathbf{A} = \begin{pmatrix}
1 & 1 & \ldots & 1 & 0 & 0 & \ldots & 0 & \ldots & 0 & 0 & \ldots & 0 \\
0 & 0 & \ldots & 0 & 1 & 1 & \ldots & 1 & \ldots & 0 & 0 & \ldots & 0 \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & \ldots & 1 & 1 & \ldots & 1 \\
1 & 0 & \ldots & 0 & 1 & 0 & \ldots & 0 & \ldots & 1 & 0 & \ldots & 0 \\
0 & 1 & \ldots & 0 & 0 & 1 & \ldots & 0 & \ldots & 0 & 1 & \ldots & 0 \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
0 & 0 & \ldots & 1 & 0 & 0 & \ldots & 1 & \ldots & 0 & 0 & \ldots & 1
\end{pmatrix}
$$

Ova matrica je formata $(m+n) \times m\,n$. Recimo, za $m = n = 3$ dobija se matrica

$$
\mathbf{A} = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1
\end{pmatrix}
$$

Odmah se vidi da su gore navedeni uvjeti za totalnu unimodularnost ispunjeni ukoliko uzmemo da skup P sadrži prvih m redova matrice **A**, a skup Q preostalih n redova. Slijedi da je matrica **A** *totalno unimodularna*. Ovo objašnjava zašto transportni problemi uvijek imaju cjelobrojna optimalna rješenja kad god je vektor **b** cjelobrojan. Doduše, matrica **A** nema puni rang $m + n$, nego joj je rang $m + n - 1$. Međutim, kako se jedno od ograničenja u transportnom problemu uvijek može izostaviti jer je posljedica ostalih, iz matrice A možemo *izbaciti jedan red* (recimo posljednji) i na taj način dobiti *totalno unimodularnu matricu punog ranga*.

Mađarski algoritam

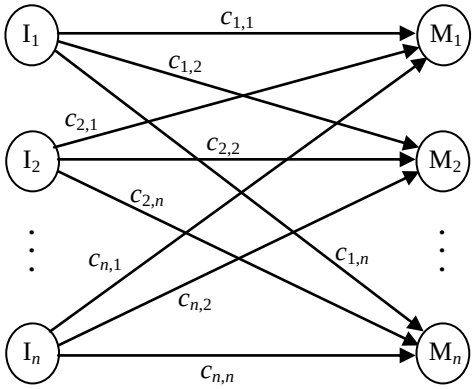
Sada ćemo preći na prikaz jednog od najpoznatijih algoritama za rješavanje problema raspoređivanja, poznatog pod nazivom ***mađarski algoritam***. Ovaj algoritam nastao je *nadgradnjom* jedne intuitivne i vrlo jednostavne tehnike poznate kao ***tehnika Flooda*** koja, sama za sebe, u jednostavnijim primjerima uspijeva da pronađe optimalno rješenje problema raspoređivanja, ali u tome *ne uspijeva uvijek*. Zapravo, tehnika Flooda se tipično koristi kao *pripremna faza* u izvođenju mađarskog algoritma.

Obično se smatra da je tvorac mađarskog algoritma *H. Kuhn* koji ga je publikovao 1955. godine, pri čemu mu je dao ime "mađarski algoritam" zbog činjenice da je ključne ideje za razvoj algoritma dobio iz nekih radova mađarskih matematičara *D. Kőniga* i *J. Egerváryja* iz oblasti teorije grafova (interesantno je napomenuti da je Kuhn za potrebe razvoja ovog algoritma prevodio neke Egerváryjeve radove pisane na mađarskom jeziku *samo uz pomoć mađarsko-engleskog riječnika, ne znajući pri tome ništa o mađarskom jeziku*). Kuhn nije znao ništa o perfomansama svog algoritma, samo je dokazao da on *uvijek završava nakon konačno mnogo iteracija*. Ubrzo nakon njegove publikacije, *J. Munkres* je dokazao da je ovaj algoritam zapravo *vrlo efikasan* i pokazao kako se može dodatno ubrzati sa faktorom n u odnosu na naivnu izvedbu. Zbog toga se ovaj algoritam često naziva i ***Kuhn-Munkresov algoritam***. Međutim, nedavno je otkriveno da je ovaj algoritam znao još *C. G. Jacobi* u 19-tom vijeku.

Za razliku od metoda za rješavanje transportnog problema, koji su u osnovi zasnovani na simpleks metodu, te spadaju u *primalne metode* (jer do rješenja dolaze kroz niz iteracija koje generiraju *primalno dopustiva rješenja*), mađarski algoritam je, u osnovi, *dualni metod*. To znači da ovaj algoritam, slično *dualnom simpleks metodu*, kroz iteracije generira rješenja koja *nisu primalno ali jesu dualno dopustiva*, sve dok se ne dostigne primalna dopustivost (a samim tim i optimalnost, kao što slijedi iz opće teorije dualnosti). Ipak, mađarski algoritam nije specijalna forma dualnog simpleks metoda, jer se prelazak sa jednog na drugo dualno dopustivo rješenje ne vrši klasičnim putem kao kod dualnog simpleks metoda, nego se za tu svrhu koriste *neke ideje iz teorije grafova*. Kasnije se pokazalo da se te ideje iz teorije grafova mogu interpretirati kao *rješavanje parova nekih pomoćnih jednostavnih problema linearnog programiranja* koji su jedan drugom *međusobno dualni*, tako da se može reći da mađarski algoritam spada u porodicu tzv. ***primalno-dualnih metoda***, u koje, pored njega, spada još veliki broj raznih drugih algoritama koji se koriste u kombinatornoj optimizaciji (možda najpoznatiji primjer je *Dijkstrin algoritam* za nalaženje najkraćeg puta između zadana dva čvora u grafu). Primalno-dualni metodi su danas najčešće korištena strategija za *rješavanje problema kombinatorne optimizacije*.

Mađarski algoritam se u raznoj literaturi susreće u brojnim prividno različitim ali zapravo ekvivalentnim verzijama. Sve te verzije se mogu svrstati u dvije grupe: *matrične verzije* i *grafovske verzije* (iskazane *jezikom teorije grafova*). Matrične verzije su historijski starije, i znatno su pogodnije za ručni rad od grafovskih verzija. Zapravo, matrične verzije su iskazane u formi koja je *namjerno optimizirana* za ručni rad (s obzirom da su u doba kada je Kuhn publikovao ovaj algoritam računari bili više kuriozitet nego nešto što je u masovnoj upotrebi). S druge strane, grafovske verzije su prilično *nepregledne za ručni rad*, ali daju bolji uvid u *samu prirodu metoda* i ukazuju na to kako se implementacija algoritma može ubrzati sa faktorom n, što je vrlo teško uočiti u matričnoj verziji. Razlog zbog čega je matrična verzija relativno neefikasna za implementaciju na računaru leži u činjenici da čovjek *drugačije procesira informacije* od računara (čovjek koristi *veoma veliki stepen paralelizma*, dok je računar *dominantno sekvencijalna mašina*), tako da su neke operacije koje su za čovjeka trivijalne, vremenski zahtjevne za izvođenje na računaru. Recimo, neka je potrebno locirati *sve elemente u matrici koji su jednaki nuli*. Čovjek odgovor na to pitanje daje praktično *trenutno*, prostim *pogledom* na matricu (ukoliko matrica nije prevelika), dok je računaru potrebno n^2 testova (za matrice formata $n \times n$) da bi odgovorio na ovo pitanje, čak i u slučaju kada postoji samo nekoliko elemenata koji su jednaki nuli.

Matrične verzije algoritma vrše manipulacije nad *matricom efikasnosti* **C** čiji su elementi koeficijenti funkcije cilja $c_{i,j}$, $i = 1 \,..\, n$, $j = 1 \,..\, n$. Ova matrica se često predstavlja u vidu *tablice*, tako da se manipulacije vrše nad elementima tablice. S druge strane, grafovske verzije posmatraju problem raspoređivanja kao *grafovski problem*. Zaista, problem raspoređivanja se može modelirati sljedećim težinskim grafom:



Ovaj graf je vrlo posebnog oblika. Svi njegovi čvorovi se mogu podijeliti u *dvije porodice* čvorova (I_i, $i = 1 .. n$ i M_j, $j = 1 .. n$) koji su takvi da je *svaki čvor iz prve porodice spojen granom sa svakim čvorom iz druge porodice*, dok *nijedna dva čvora iz iste porodice nisu spojeni granom*. Za takve grafove kažemo da su ***bikompletni***. Sada se problem raspoređivanja jezikom teorije grafova može iskazati kao problem nalaženja ***savršenog uparivanja minimalne ukupne težine*** u ovom grafu, odnosno takvog skupa grana da je *svaki čvor grafa krâj tačno jedne grane iz tog skupa*, i da pri tome, *suma težina svih grana u tom skupu bude što je god moguće manja*.

U nastavku ćemo se pretežno fokusirati na *matričnu formulaciju* algoritma, s obzirom da je pogodnija za ručni rad, uz *povremene osvrte* na grafovsku formulaciju. Interesantno je da je literatura često prepuna raznih *nekompletnih* (pa čak i *netačnih*) opisa matrične formulacije mađarskog algoritma koji *ne rade* u svim slučajevima (odnosno kod kojih se dešava da se ponekad neki koraci algoritma prosto *ne mogu primijeniti*), ili opisa u kojima je *nejasno* kako konkretno provesti neke korake algoritma u iole složenijim slučajevima (da ne govorimo o tome kako te korake *isprogramirati*). Verzija koja će biti ovdje prikazana je *kompletna* i *nedvosmislena*. S druge strane, grafovske formulacije koje se u literaturi susreću su gotovo uvijek korektne i precizne.

Za opis matrične formulacije algoritma potrebno je definirati pojam ***skupa nezavisnih nula*** u matrici. Za neki skup elemenata u matrici koji su jednaki nuli kažemo da je *skup nezavisnih nula* ukoliko svaki red i svaka kolona matrice sadrži *tačno jedan element* iz tog skupa. Elementi tog skupa se tada nazivaju ***nezavisne nule***, dok se ostali elementi jednaki nuli u matrici nazivaju ***zavisne nule*** (jasno je da razvrstavanje nula na zavisne i nezavisne *nije jedinstveno*). Za neki skup nezavisnih nula se kaže da je ***maksimalan skup nezavisnih nula*** ukoliko ne postoji drugi skup nezavisnih nula koji sadrži više nezavisnih nula od njega. Nakon uvođenja ovih pojmova, kostur mađarskog algoritma moguće je iskazati na sljedeći način (primijetimo da *nije na prvi pogled jasno* kako se provode koraci 3 i 4 u ovom kosturu algoritma, mada se oni pri ručnom radu kod tablica manjih dimenzija veoma lako izvode čistim *isprobavanjem*):

1. Od *svih elemenata svakog reda* tablice *oduzima se najmanji element tog reda* (na taj način se postiže da će se *u svakom redu novodobijene tablice nalaziti barem jedna nula*).
2. Od *svih elemenata svake kolone* novodobijene tablice *oduzima se najmanji element svake kolone* (nakon ovoga će se i u *svakoj koloni novodobijene tablice nalaziti barem jedna nula*). Striktno gledano, ovaj korak *nije neophodan* u algoritmu, te ga neki autori *izostavljaju*, ali može *značajno smanjiti broj iteracija* koje su potrebne da se pronađe optimalno rješenje.
3. Pronađe se *maksimalan skup nezavisnih nula* u tablici. Neka ovaj skup sadrži $k \leq n$ nezavisnih nula. Ukoliko je $k = n$, prelazimo na korak 7.
4. Precrtaju se *redovi i kolone* tablice *horizontalnim i vertikalnim linijama* tako da se upotrijebi *što je god moguće manje linija*, a da pri tome budu precrtana *sva polja koja su jednaka nuli*. Ukoliko je nađeni skup nezavisnih nula *zaista maksimalan*, ovo bi trebalo biti moguće izvesti sa *tačno k linija*. Ukoliko se ispostavi da to nikako nije moguće obaviti pomoću k linija, to znači da nađeni skup nezavisnih nula *nije maksimalan*. Tada je potrebno *korigovati* nađeni skup nezavisnih nula sa ciljem da se pronađe *veći* skup nezavisnih nula. Kada se takav skup pronađe, vraćamo se ponovo na korak 4, *osim ukoliko je pronađen skup koji sadrži tačno n nezavisnih nula* (tada prelazimo na korak 7).
5. Pronađemo *najmanji element* Δ u tablici koji je ostao *neprecrtan* nakon koraka 4.

6. Oduzmemo Δ od svih elemenata tablice koji su ostali *neprecrtani*, a dodamo Δ na sve elemente tablice koji su precrtani *i horizontalnom i vertikalnom linijom*. Tehnički, ovo je ekvivalentno oduzimanju Δ od elemenata *svih redova koji nisu precrtani*, a nakon toga dodavanju Δ na elemente *svih precrtanih kolona*. Nakon obavljene transformacije vraćamo se na korak 3.
7. Optimalno rješenje je *nađeno* i očitava se na sljedeći način. Za sve $i = 1 .. n$ i $j = 1 ... n$ ukoliko se na poziciji (i, j) u tablici nalazi *nezavisna nula*, tada je $x_{i,j} = 1$, inače je $x_{i,j} = 0$.

Koraci 1, 2 i 3 zapravo predstavljaju već spomenutu *tehniku Flooda*. Često se dešava da se već nakon koraka 3 dobija optimalno rješenje. Mađarski algoritam je *proširenje* tehnike Flooda za slučajeve kada u koraku 3 nije dobijeno n nezavisnih nula, odnosno kada optimalno rješenje *nije odmah nađeno*.

Treba istaći da se u većini literature često *ne navode* korekcije opisane u koraku 4 nego se *prećutno podrazumijeva* da je korak 3 obavljen kako treba, odnosno da je u koraku 3 uvijek pronađen zaista *maksimalan* skup nezavisnin nula. Međutim, u praksi su ove korekcije *itekako bitne*, pogotovo u *efikasnim implementacijama* mađarskog algoritma (koje se ne trude da po svaku cijenu u koraku 3 *garantirano* pronađu maksimalan skup nezavisnih nula).

Razmotrimo sada prvo kako se izvodi korak 3. Iskazano jezikom teorije grafova, ovaj korak je zapravo pronalaženje *maksimalnog uparivanja* u jednom *podgrafu* polaznog grafa koji se sastoji *samo od onih grana polaznog grafa koje odgovaraju nulama u transformiranoj tablici* koja je dobijena nakon koraka 1 i 2. Te grane se često nazivaju ***zategnute grane*** (engl. *tight edges*), tako da se često govori o ***podgrafu zategnutih grana*** ili ***podgrafu jednakosti*** (ovaj drugi naziv vezan je za *dualnu interpretaciju*, o kojoj ćemo nešto kasnije govoriti). Ovi podgrafovi su naravno *bipartitivni* jer je polazni graf bio *bikompetan*. Za nalaženje maksimalnog uparivanja u bipartitivnim grafovima u teoriji grafova ima mnogo lijepih algoritama, recimo *Ford-Fulkersonov algoritam* (specijalan slučaj njihovog algoritma za nalaženje *maksimalnog protoka u transportnim mrežama*), te nešto brži *Hopcroft-Karpov algoritam*. Ova dva algoritma se, u najgorem slučaju, izvršavaju u vremenu proporcionalnom sa n^3 odnosno $n^{2,5}$. Međutim, u praksi se za izvođenje koraka 3 obično primjenjuje sljedeći *pohlepni algoritam* koji doduše *ne garantira* pronalaženje *maksimalnog* skupa nezavisnih nula (zato će ponekad biti potrebne *korekcije* spomenute na kraju koraka 4), ali je *vrlo jednostavan za izvođenje* i ne traži više od n^2 operacija:

3.1. Na početku postupka, sve nule u tablici su *nerazvrstane* (tj. nisu svrstane ni u zavisne, ni u nezavisne).
3.2. Ukoliko u nekom *redu* ili nekoj *koloni* tablice postoji samo *jedna nerazvrstana nula*, tu nulu proglasimo za *nezavisnu nulu*, a sve ostale nule u odgovarajućoj *koloni* odnosno *redu* proglasimo za *zavisne nule*. Pri ručnom radu, obično se nezavisne nule *zaokružuju*, a zavisne *precrtavaju*. Ukoliko niti u jednom redu i niti u jednoj koloni *ne postoji* samo jedna nerazvrstana nula, tada se *proizvoljno* proglašava jedna od nula za nezavisnu, dok se sve ostale nule u *istom redu i koloni* proglašavaju za zavisne. Obično se izbor vrši iz onog reda ili kolone u kojem se nalazi *najmanji broj nerazvrstanih nula*.
3.3. Ukoliko je preostalo još nerazvrstanih nula, vraćamo se na korak 3.2. U suprotnom, postupak se završava.

Ukoliko prilikom primjene ovog postupka *nikada nismo imali dilemu* koju nulu proglasiti za nezavisnu, odnosno ukoliko smo tokom cijelog postupka uvijek imali samo jednu nerazvrstanu nulu u makar jednom redu i koloni, tada je pronađeni skup nezavisnih nula *garantirano maksimalan*. U suprotnom, nađeni skup nezavisnih nula *može ali i ne mora biti maksimalan*. Situacija može ovisiti od toga *koje smo nule proglašavali za nezavisne u slučajevima kada izbor nije bio jednoznačan*. Inače, nađenim zavisnim i nezavisnim nulama odgovaraju *uparene i neuparene grane* u grafovskoj terminologiji.

Razmotrimo sada kako se izvodi korak 4. Izraženo jezikom teorije grafova, ovaj korak zapravo nalazi ***minimalni pokrivač vrhova*** (engl. *minimum vertex cover*) za podgraf zategnutih grana (minimalni pokrivač vrhova je najmanji skup čvorova takav da svaka grana grafa ima krâj u nekom elementu tog skupa). Inače, interesantno je da se i problem maksimalnog uparivanja i problem minimalnog pokrivača vrhova mogu iskazati kao problemi *linearnog programiranja* i da su oni *međusobno dualni* (u ovome se ogleda *primalno-dualna priroda* mađarskog algoritma, koji je u osnovi dualni metod). U mađarskom algoritmu, ovaj korak se izvodi postupkom koji slijedi iz jednog teorema iz teorije grafova, koji je poznat kao ***Kőnigov*** (ili ***Kőnig-Egerváryjev***) ***teorem***. Ovaj postupak je, u suštini, *varijanta BFS pretrage* koja naizmjenično posjećuje zavisne i nezavisne nule u tablici, krećući se naizmjenično horizontalno i vertikalno, koja polazi od redova u kojima *nema nezavisnih nula* (odnosno od *neuparenih vrhova* u grafovskoj terminologiji). Konkretno taj postupak teče ovako:

4.1. *Označimo* na neki način (recimo zvjezdicama ili strelicama) *redove koji ne sadrže nezavisne nule*.
4.2. *Označimo kolone* koje u *upravo označenim redovima imaju zavisne nule*, a koje *ranije nisu bile označene*. Ukoliko takvih kolona *nema*, prelazimo na korak 4.6.
4.3. Ukoliko neka od upravo označenih kolona *ne sadrži niti jednu nezavisnu* nulu, prelazimo na korak 4.5.
4.4. Označimo *redove* koji imaju *nezavisnu nulu* u *upravo označenim kolonama* i vraćamo se na korak 4.2.
4.5. Ukoliko smo došli do ove tačke, to znači da postupak pod 3 nije obavljen korektno, odnosno da nije pronađen *maksimalan* skup nezavisnih nula, te je potrebno obaviti *korekciju* skupa nezavisnih nula (koju ćemo detaljno opisati u nastavku), nakon čega se korak 4 *završava*.
4.6. Precrtamo sve *neoznačene redove* i sve *označene kolone* i postupak je *gotov*. Kako se svaka označena kolona uvijek *precrtava* na kraju, neki autori ne označavaju kolone, nego svaku kolonu odmah *precrtaju* čim dođe red na nju da se označi.

Još je potrebno objasniti kako se vrši korekcija u koraku 4.5. Pri ručnom radu sa problemima raspoređivanja manjih dimenzija, ove korekcije su prilično rijetko potrebne, jer se *obično u koraku* 3 *pronađe maksimalan skup nezavisnih nula*. Međutim, u problemima većih dimenzija nekada se pri ručnom radu desi da u koraku 3 ne bude pronađen maksimalan skup nezavisnih nula, tako da su korekcije potrebne. Isto tako, korekcije se *redovno vrše* u efikasnim implementacijama mađarskog računara na računaru. Objasnimo stoga kako se ove korekcije vrše. Ukoliko smo uopće došli na podkorak 4.5, to zapravo znači da smo *pronašli put* kroz tablicu koji ide *naizmjenično horizontalno i vertikalno* i *naizmjenično posjećuje zavisne i nezavisne nule*, a počinje u *koloni u kojoj nema nezavisnih nula* i završava u *redu u kojem nema nezavisnih nula*. Iskazano jezikom teorije grafova, to je put koji *naizmjenično prolazi kroz uparene i neuparene grane*, a *počinje i završava u neuparenim čvorovima* (u teoriji grafova, takav put se zove ***povećavajući put***). Ukoliko duž takvog puta *svaku zavisnu nulu proglasimo za nezavisnu a svaku nezavisnu nulu za zavisnu*, dobićemo novi skup nezavisnih nula koji ima *jednu nezavisnu nulu više* u odnosu na prethodni skup. Na taj način smo izvršili *povećanje* skupa nezavisnih nula. Ostaje još pitanje *kako najlakše očitati* taj povećavajući put. To je jednostavno obaviti prateći *unazad* redoslijed kojim su označavane kolone i redovi. Da bi se lakše snašli, moguće je redove i kolone umjesto zvjezdicama ili strelicama označavati *rednim brojevima* od 1 naviše, pri čemu se sa *k* označavaju oni redovi i kolone koji su označeni u *k*-toj "turi" označavanja. Tada se povećavajući put lako očitava prateći brojeve *unazad*, naizmjenično gledajući kolone i redove.

Ovim je završen kompletan opis mađarskog algoritma u *matričnoj formi*, uz povremene osvrte na neke grafovske interpretacije. Mada kompletan postupak djeluje pomalo komplicirano, mađarski algoritam je *izuzetno jednostavan* za ručni rad (dok računarska implementacija zahtijeva *izvjesno umijeće*, pogotovo ukoliko želimo da implementacija bude *efikasna*). Sve će biti mnogo jasnije nakon što se razmotri nekoliko konkretnih primjera.

- ***Primjer***: Primjenom mađarskog algoritma, naći optimalan raspored pet izvršilaca $I_1 - I_5$ na pet radnih mjesta $M_1 - M_5$, pri čemu su troškovi koji se postižu raspoređivanjem svakog od kandidata na svako od raspoloživih radnih mjesta dati u sljedećoj tabeli:

	M_1	M_2	M_3	M_4	M_5
I_1	11	17	8	16	20
I_2	9	7	12	6	15
I_3	13	16	15	12	16
I_4	21	24	17	28	26
I_5	14	10	12	11	15

U prvom koraku, od svih članova svakog reda *oduzima se vrijednost najmanjeg elementa tog reda*. Najmanji elementi redova $I_1 - I_5$ su respektivno 8, 6, 12, 17 i 10, tako da nakon ovog koraka dobijamo sljedeću tablicu:

	M_1	M_2	M_3	M_4	M_5
I_1	3	9	0	8	12
I_2	3	1	6	0	9
I_3	1	4	3	0	4
I_4	4	7	0	11	9
I_5	4	0	2	1	5

U drugom koraku, od svih članova svake kolone *oduzima se vrijednost najmanjeg elementa te kolone*. Najmanji elementi kolona $M_1 - M_5$ su respektivno 1, 0, 0, 0 i 4, tako da nakon ovog koraka dobijamo sljedeću tablicu:

	M_1	M_2	M_3	M_4	M_5
I_1	2	9	0	8	8
I_2	2	1	6	0	5
I_3	0	4	3	0	0
I_4	3	7	0	11	5
I_5	3	0	2	1	1

Sada je potrebno *razvrstati* sve nule u tablici na *zavisne* i *nezavisne*. Primijenićemo opisani *pohlepni postupak*, pri čemu ćemo nezavisne nule označavati znakom "●", a zavisne nule znakom "∅". Vidimo da u redu I_1 imamo samo jednu nerazvrstanu nulu na poziciji $(1, 3)$, pa ćemo je proglasiti za nezavisnu, a ostale nule u odgovarajućoj koloni (M_3) proglasićeno za zavisne. To je samo nula na poziciji $(4, 3)$. Red I_2 također sadrži samo jednu nerazvrstanu nulu na poziciji $(2, 4)$, koja postaje nezavisna nula, dok nula na poziciji $(3, 4)$ u istoj koloni postaje zavisna. Sljedeći red koji sadrži samo jednu nerazvrstanu nulu je I_5, tako da nula na poziciji $(5, 2)$ postaje nezavisna (kako kolona M_2 ne sadrži nule, neće biti novih zavisnih nula u ovom koraku). Dalje, imamo samo jednu nerazvrstanu nulu u koloni M_1, tako da ta nula postaje nezavisna, dok ostale nule u odgovarajućem redu (I_3) postaju zavisne. Ovim smo razvrstali sve nule, tako da dobijamo situaciju prikazanu u sljedećoj tabeli:

	M_1	M_2	M_3	M_4	M_5
I_1	2	9	●	8	8
I_2	2	1	6	●	5
I_3	●	4	3	∅	∅
I_4	3	7	∅	11	5
I_5	3	●	2	1	1

Kako tokom postupka nigdje nismo imali dilemu po pitanju koju nulu izabrati, nađeni skup nezavisnih nula je *sigurno maksimalan*. Ovim je zapravo okončana *tehnika Flooda*. Međutim, optimalno rješenje *nije nađeno*, jer nam *nedostaje jedna nezavisna nula* (trebalo bi nam 5 nezavisnih nula za optimalno rješenje). Slijedi da moramo *nastaviti postupak*.

Sada je, prema koraku 4, potrebno *precrtati sve nule* u tablici sa 4 linije. Prostim gledanjem nije teško vidjeti da je to moguće uraditi recimo tako što ćemo precrtati redove I_2, I_3, I_5 i kolonu M_3 (ovo *nije jedina mogućnost*). Međutim, ovdje ćemo, radi ilustracije, primijeniti opisani *formalni postupak* zasnovan na BFS pretrazi. Stoga ćemo prvo označiti (recimo strelicom) *redove koji nemaju nezavisnih nula*. Takav je samo red I_4. Zatim ćemo označiti kolone koje u *označenom redu imaju zavisne nule*. Takva je samo kolona M_3. Ovim dobijamo sljedeću situaciju:

	M_1	M_2	M_3	M_4	M_5	
I_1	2	9	●	8	8	
I_2	2	1	6	●	5	
I_3	●	4	3	∅	∅	
I_4	3	7	∅	11	5	←
I_5	3	●	2	1	1	
			↑			

Dalje, treba označiti redove koje imaju nezavisnu nulu u upravo označenim kolonama (tj. koloni M_3). Takav je samo red I_1. Sada bi trebalo označiti sve kolone koje u redu I_1 imaju zavisne nule. Međutim, kako takvih kolona *nema*, jedino što nam je ostalo je da *precrtamo sve neoznačene redove i sve označene kolone* i postupak je *gotov*. Time dobijamo situaciju prikazanu na sljedećoj slici:

	M_1	M_2	M_3	M_4	M_5	
I_1	2	9	•	8	8	←
~~I_2~~	~~2~~	~~1~~	~~6~~	•	~~5~~	
~~I_3~~	•	~~4~~	~~3~~	∅	∅	
I_4	3	7	∅	11	5	←
~~I_5~~	~~3~~	•	~~2~~	~~1~~	~~1~~	
			↑			

Sada pronalazimo *najmanji neprecrtani element*, koji iznosi $\Delta = 2$ (na poziciji $(1,1)$). Njega *oduzimamo* od svih *neprecrtanih elemenata*, a *dodajemo* na *dvostruko precrtane elemente*, tj. elemente na pozicijama $(2,3)$, $(3,3)$ i $(5,3)$ (ovo je *istovjetno* kao *oduzimanje* Δ od *svih elemenata u neprecrtanim kolonama*, a nakon toga *dodavanje* Δ na *sve elemente u precrtanim kolonama*). Nakon ovoga, dobija se sljedeća tablica:

	M_1	M_2	M_3	M_4	M_5
I_1	0	7	0	6	6
I_2	2	1	8	0	5
I_3	0	4	5	0	0
I_4	1	5	0	9	3
I_5	3	0	4	1	1

Sada ponovo treba izvršiti razvrstavaje nula na zavisne i nezavisne. Prvo ćemo za nezavisnu nulu proglasiti nulu na poziciji $(4,3)$, jer je *jedina nerazvrstana* u svom redu. Time nula na poziciji $(1,3)$ postaje zavisna. Sada možemo za nezavisnu nulu proglasiti nulu na poziciji $(1,1)$, jer je sad ona ostala jedina nerazvrstana. S obzirom da ćemo u daljem toku postupka uvijek imati barem ponegdje samo jednu nerazvrstanu nulu, nećemo prikazivati sve detalje postupka. Sam izbor nezavisnih nula može se vršiti *različitim redoslijedom*, ali bez obzira na redoslijed, *uvijek ćemo doći do sljedeće situacije*:

	M_1	M_2	M_3	M_4	M_5
I_1	•	7	∅	6	6
I_2	2	1	8	•	5
I_3	∅	4	5	∅	•
I_4	1	5	•	9	3
I_5	3	•	4	1	1

Kako sada imamo tačno 5 nezavisnih nula, *pronađeno je optimalno rješenje*. Ovo rješenje glasi $x_{1,1} = x_{2,4} = x_{3,5} = x_{4,3} = x_{5,2} = 1$, dok su ostale promjenjive jednake nuli. Drugim riječima optimalan raspored izvršilaca na radna mjesta je $I_1 \to M_1$, $I_2 \to M_4$, $I_3 \to M_5$, $I_4 \to M_3$ i $I_5 \to M_2$. Optimalna vrijednost funkcije cilja iznosi $Z = c_{1,1} + c_{2,4} + c_{3,5} + c_{4,3} + c_{5,2} = 11 + 6 + 16 + 17 + 10 = 60$. Pri tome, optimalna vrijednost funkcije cilja može se izračunati *samo na osnovu početne* (*neizmijenjene*) *tablice*.

- ***Primjer***: Neka radna organizacija treba da otvori četiri radna mjesta $R_1 - R_4$. Raspisan je konkurs na koji se prijavilo 5 kandidata $K_1 - K_5$. Obavljeno je testiranje njihove efikasnosti (stručnosti) za svako od ponuđenih radnih mjesta, a rezultati testiranja su prikazani u sljedećoj tabeli. Odrediti kako treba optimalno rasporediti prijavljene kandidate na radna mjesta da se pri tome postigne što je god moguće veća ukupna efikasnost.

	R_1	R_2	R_3	R_4
K_1	5	6	5	1
K_2	4	6	4	1
K_3	8	6	7	6
K_4	2	4	4	4
K_5	6	10	9	4

U ovom primjeru, radi se o *nebalansiranom problemu raspoređivanja*, jer je $m = 5$ i $n = 4$. Stoga je neophodno tablicu proširiti *dodatnom petom kolonom*, čije ćemo elemente *postaviti na nulu*. Ovo proširivanje efektivno predstavlja uvođenje *fiktivnog, petog radnog mjesta* R_5 za koje *niti jedan kandidat nije podoban* (tako da će na to nepostojeće radno mjesto biti raspoređen *samo onaj kandidat za kojeg se ispostavi da nije korisno da bude raspoređen niti na jedno konkretno radno mjesto*). Dalje, kako je očito u pitanju problem *maksimizacije, negiraćemo sve elemente u tablici*. Ovim se problem *svodi na ekvivalentni problem minimizacije*, kojem odgovara sljedeća tablica:

	R_1	R_2	R_3	R_4	R_5
K_1	–5	–6	–5	–1	0
K_2	–4	–6	–4	–1	0
K_3	–8	–6	–7	–6	0
K_4	–2	–4	–4	–4	0
K_5	–6	–10	–9	–4	0

Vrijedi napomenuti da se tehnika Flooda ponekad provodi *obrnutim redoslijedom*, odnosno prvo se vrši redukcija *kolona*, pa tek onda *redova* (tj. redoslijed prva dva koraka se *obrće*). Na taj način se dobija *drugačija reducirana matrica* (pa se samim tim i tok narednih iteracija *razlikuje*), ali se na kraju mora doći do *istog optimalnog rješenja* (osim možda ako nije jedinstveno). Radi promjene, ovaj primjer ćemo uraditi upravo na taj način. Dakle, prvo ćemo reducirati kolone.Najmanji elementi kolona $R_1 - R_5$ su –8, –10, –9, –6 i 0 respektivno. Stoga od svih elemenata kolone R_1 oduzimamo broj –8 (odnosno dodajemo broj 8), od svih elemenata kolone R_2 oduzimamo broj –10, itd. Time dolazimo do sljedeće tablice:

	R_1	R_2	R_3	R_4	R_5
K_1	3	4	4	5	0
K_2	4	4	5	5	0
K_3	0	4	2	0	0
K_4	6	6	5	2	0
K_5	2	0	0	2	0

U sljedećem koraku bismo trebali od svih elemenata svakog reda oduzeti najmanji element tog reda. Međutim, kako je najmanji element u svakom od redova nula, ovaj korak ne dovodi ni do kakve promjene gore prikazane tablice. Stoga prelazimo na nalaženje maksimalnog skupa nezavisnih nula. Kako se u ovom primjeru u ovom koraku ne javljaju nikakve nedoumice, nećemo prikazivati sve detalje postupka, nego ćemo samo prikazati rezultat obavljenog razvrstavanja:

	R_1	R_2	R_3	R_4	R_5
K_1	3	4	4	5	●
K_2	4	4	5	5	∅
K_3	●	4	2	∅	∅
K_4	6	6	5	2	∅
K_5	2	●	∅	2	∅

Kako nađeni skup nezavisnih nula ima samo tri nule, moramo preći na etapu *precrtavanja*, odnosno trebamo sve nule u tablici precrtati sa *tri linije*. Lako se vidi da to možemo uraditi provlačenjem linije kroz redove K_3 i K_5, te kolonu R_5. Ukoliko to nismo odmah vidjeli, uvijek možemo postupiti formalnim postupkom. Prvo označimo redove K_2 i K_4 u kojima se ne nalaze nezavisne nule. Zatim označavamo kolone koje imaju zavisne nule u ovim redovima. Takva je samo kolona R_5. Nakon toga, označavamo red K_1 koji ima nezavisnu nulu u upravo označenoj koloni R_5. Označavanje se ne može dalje nastaviti, jer red K_1 ne sadrži zavisne nule. Stoga ćemo precrtati sve neoznačene redove (K_3 i K_5) te označenu kolonu R_5 i postupak precrtavanja je završen. Situacija je sada kao na sljedećoj slici:

	R_1	R_2	R_3	R_4	R_5	
K_1	3	4	4	5	•	←
K_2	4	4	5	5	∅	←
~~K_3~~	~~•~~	~~4~~	~~2~~	~~∅~~	~~∅~~	
K_4	6	6	5	2	∅	←
~~K_5~~	~~2~~	~~•~~	~~0~~	~~2~~	~~∅~~	
			↑			

Najmanji neprecrtani element je 2, u presjeku reda K_4 i kolone R_4. Stoga vrijednost 2 oduzimamo od svih neprecrtanih elemenata, a dodajemo na sve elemente koji su precrtani i horizontalnom i vertikalnom linijom (u našem slučaju, to su elementi koji se nalaze u presjeku redova K_3 i K_5 sa kolonom R_5). Nakon ovoga, dobijamo sljedeću tablicu:

	R_1	R_2	R_3	R_4	R_5
K_1	1	2	2	3	0
K_2	2	2	3	3	0
K_3	0	4	2	0	2
K_4	4	4	3	0	0
K_5	2	0	0	2	2

Vraćamo se na razvrstavanje nula. Ni u ovom slučaju nema nikakvih nedoumica kako to razvrstavanje obaviti, pa nećemo prikazivati detalje postupka. Nakon razvrstavanja dobija se situacija kao u sljedećoj tablici:

	R_1	R_2	R_3	R_4	R_5
K_1	1	2	2	3	•
K_2	2	2	3	3	∅
K_3	•	4	2	∅	2
K_4	4	4	3	•	∅
K_5	2	•	∅	2	2

Ni ovaj put nemamo dovoljno nezavisnih nula da bismo mogli očitati optimalno rješenje. Nastavljamo sa etapom precrtavanja. Precrtavanje svih nula sa četiri linije možemo postići ukoliko precrtamo redove K_3, K_4 i K_5, te kolonu R_5. Formalno, do tog zaključka možemo doći ovako. Jedini red koji ne sadrži nezavisnu nulu je K_2, te ćemo ga označiti. Nakon toga označavamo kolonu R_5, jer jedino ona sadrži zavisnu nulu u upravo označenom redu K_2. Nakon toga ćemo označiti one redove koji imaju nezavisnu nulu u upravo precrtanoj koloni R_5. U našem slučaju to je red K_1. Kako ovaj red ne sadrži zavisne nule, označavanje se ne može nastaviti, te precrtavamo preostale neoznačene redove K_3, K_4 i K_5 i označenu kolonu R_5. Ovim je postupak precrtavanja završen i dobijamo situaciju kao na sljedećoj slici:

	R_1	R_2	R_3	R_4	R_5	
K_1	1	2	2	3	•	←
K_2	2	2	3	3	∅	←
~~K_3~~	~~•~~	~~4~~	~~2~~	~~∅~~	~~2~~	
~~K_4~~	~~4~~	~~4~~	~~3~~	~~•~~	~~0~~	
~~K_5~~	~~2~~	~~•~~	~~∅~~	~~2~~	~~2~~	
				↑		

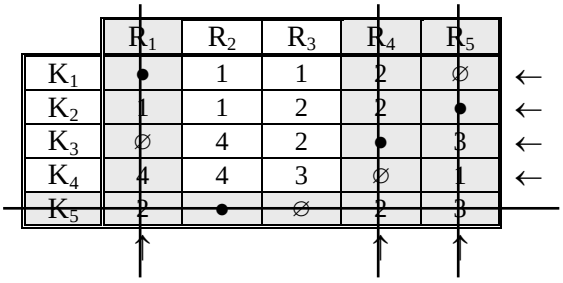
Sada je najmanji neprecrtani element 1, u presjeku reda K_4 i kolone R_1. Stoga vrijednost 1 oduzimamo od svih neprecrtanih elemenata, a dodajemo na sve elemente koji su precrtani i horizontalnom i vertikalnom linijom (u našem slučaju, to su elementi koji se nalaze u presjeku redova K_3, K_4 i K_5 sa kolonom R_5). Tako dobijamo sljedeću tablicu:

	R_1	R_2	R_3	R_4	R_5
K_1	0	1	1	2	0
K_2	1	1	2	2	0
K_3	0	4	2	0	3
K_4	4	4	3	0	1
K_5	2	0	0	2	3

Vraćamo se na razvrstavanje nula. Ponovo nećemo imati nikakvih nedoumica u izvođenju postupka i ponovo nećemo dobiti dovoljan broj nezavisnih nula da bismo mogli očitati optimalno rješenje. Situacija koja se dobije nakon razvrstavanja prikazana je u sljedećoj tablici:

	R_1	R_2	R_3	R_4	R_5
K_1	●	1	1	2	∅
K_2	1	1	2	2	●
K_3	∅	4	2	●	3
K_4	4	4	3	∅	1
K_5	2	●	∅	2	3

Nastavljamo sa etapom precrtavanja. Traženo precrtavanje (sa 4 linije) možemo dobiti ukoliko precrtamo red K_5 i kolone R_1, R_4 i R_5. Formalno, ovo precrtavanje možemo pronaći ovako. Prvo ćemo označiti red K_4, koji je jedini koji ne sadrži nezavisnu nulu. Ovaj red ima zavisnu nulu u koloni R_4, koju ćemo označiti. Označena kolona ima nezavisnu nulu u redu K_3, tako da sada označavamo ovaj red. On sadrži zavisnu nulu u koloni R_1, koju označavamo. Sada označavamo red K_1, jer upravo označena kolona ima nezavisnu nulu u tom redu. Taj red posjeduje zavisnu nulu u koloni R_5, te označavamo i ovu kolonu. Ona sadrži nezavisnu nulu u redu K_2, koji ćemo označiti. Označavanje se ovdje prekida, jer upravo označeni red K_2 ne sadrži zavisne nule. Stoga precrtavamo jedini neoznačeni red K_5 i označene kolone R_1, R_4 i R_5. Time je postupak precrtavanja završen, nakon čega je situacija kao na sljedećoj slici:



Najmanji neprecrtani element je 1 (imamo ga na više mjesta), tako da ćemo vrijednost 1 oduzeti od svih neprecrtanih elemenata, a dodati na sve dvostruko precrtane elemente (tj. elemente u presjeku reda K_5 i kolona R_1, R_4 i R_5). Tako dobijamo situaciju prikazanu u sljedećoj tabeli:

	R_1	R_2	R_3	R_4	R_5
K_1	0	0	0	2	0
K_2	1	0	1	2	0
K_3	0	3	1	0	3
K_4	4	3	2	0	1
K_5	3	0	0	3	4

Ponovo se vraćamo na razvrstavanje nula (ovaj put ćemo *imati sreće*, odnosno *moći ćemo očitati optimalno rješenje*). Kako će se ovaj put pojaviti *izvjesne dvosmislice* tokom izvođenja postupka razvrstavanja, prikazaćemo *sve detalje* postupka.

Počinjemo sa usamljenom nerazvrstanom nulom u redu K_4 (u koloni R_4), koju ćemo proglasiti za nezavisnu, a preostale nule u koloni R_4 ćemo proglasiti za zavisne. Nakon toga, red K_3 dobija usamljenu nerazvrstanu nulu (u koloni R_1) tako da ćemo i nju proglasiti za nezavisnu, uz proglašavanje preostalih nula u koloni R_1 za zavisne. U ovom trenutku *nema više redova niti kolona koji sadrže samo po jednu nerazvrstanu nulu*. Potražimo stoga redove ili kolone koji sadrže *dvije nerazvrstane nule*. Takav je recimo red K_2 (koji sadrži nerazvrstane nule u kolonama R_2 i R_5). Odlučimo se *nasumice* za nulu koja se nalazi u

koloni R_2. Nju ćemo svrstati u nezavisne nule, a ostale nule u redu K_2 i koloni R_2 ćemo proglasiti za zavisne. Red K_5 time dobija usamljenu nerazvrstanu nulu (u koloni R_3) koju proglašavamo za nezavisnu, a ostale nule u koloni R_3 proglašavamo za zavisne. Nakon ovoga, u redu K_1 ostaje samo jedna nerazvrstana nula, koju ćemo proglasiti za nezavisnu. Ovim je postupak završen jer su sve nule razvrstane. Novodobijena situacija je prikazana na sljedećoj slici:

	R_1	R_2	R_3	R_4	R_5
K_1	∅	∅	∅	2	●
K_2	1	●	1	2	∅
K_3	●	3	1	∅	3
K_4	4	3	2	●	1
K_5	3	∅	●	3	4

Vidimo da smo našli skup od pet nezavisnih nula, te je problem *riješen*. Dakle, kandidat K_1 *neće biti primljen* (s obzirom da se raspoređuje na *fiktivno radno mjesto* R_5), dok se kandidati K_2, K_3, K_4 i K_5 raspoređuju respektivno na radna mjesta R_2, R_1, R_4 i R_3. Ukoliko nas zanima koliko iznosi ukupna efikasnost pri takvom optimalnom raspoređivanju, to moramo očitati iz početne (nemodificirane) tablice, odnosno moramo sabrati izvorne efikasnosti $c_{2,2}$, $c_{3,1}$, $c_{4,4}$ i $c_{5,3}$. Optimalna efikasnost stoga iznosi $Z = 6 + 8 + 4 + 9 = 27$.

Ovim još nije rečeno sve što se može reći o ovom primjeru. Naime, pronađeno rješenje *nije i jedino optimalno rješenje*. Uzrok tome je što smo u posljednjoj etapi pri razvrstavanju nula na nezavisne i zavisne *imali dvosmislice*. Naime, kada smo razmatrali red K_2 koji sadrži nerazvrstane nule u kolonama R_2 i R_5, umjesto nule u koloni R_2 *mogli smo se odlučiti za nulu koja se nalazi u koloni* R_5. Nakon što proglasimo tu nulu za nezavisnu, proglasili bismo preostale nule u redu K_2 i koloni R_5 za zavisne. Time *dolazimo do novih dilema*, jer ponovo *nema niti jednog reda niti kolone sa usamljenim nerazvrstanim nulama*. Recimo, u redu K_1 pri izboru nezavisne nule moramo se odlučiti za nulu u koloni R_2 ili u koloni R_3. Odlučimo li se za nulu u koloni R_2, proglasićemo preostale nule u redu K_1 i koloni R_2 za zavisne. Time će preostati samo nula u presjeku reda K_5 i kolone R_3, koju ćemo proglasiti za nezavisnu. Međutim, odlučimo li se da nezavisna bude nula u koloni R_3, proglasićemo preostale nule u redu K_1 i koloni R_3 za zavisne, nakon čega preostaje samo nula u presjeku reda K_5 i kolone R_2, koja onda postaje nezavisna. Lako se vidi da svaki drugi eventualni redoslijed izbora vodi ka jednom od ova dva rješenja. Tako, pored prethodno nađenog, postoje *još dva optimalna rješenja*, koja odgovaraju situacijama prikazanim u sljedećim tablicama:

	R_1	R_2	R_3	R_4	R_5
K_1	∅	●	∅	2	∅
K_2	1	∅	1	2	●
K_3	●	3	1	∅	3
K_4	4	3	2	●	1
K_5	3	∅	●	3	4

	R_1	R_2	R_3	R_4	R_5
K_1	∅	∅	●	2	∅
K_2	1	∅	1	2	●
K_3	●	3	1	∅	3
K_4	4	3	2	●	1
K_5	3	●	∅	3	4

Naravno, sva nađena optimalna rješenja daju istu ukupnu efikasnost (27). U svakom slučaju, nema nikakve dileme da kandidat K_3 treba biti raspoređen na radno mjesto R_1, a kandidat K_4 na radno mjesto R_4, jer sva nađena optimalna rješenja predviđaju tako. Najneizvjesnija je sudbina kandidata K_1 i K_2, tako da će radna organizacija morati pronaći dodatne kriterije (recimo, obaviti dopunska testiranja) da utvrdi da li da primi kandidata K_1 ili K_2. Pri tome, ukoliko odluka padne na kandidata K_1, biće potrebni dodatni kriteriji da se utvrdi da li je bolje da kandidat K_1 ide na radno mjesto R_2 a kandidat K_5 na radno mjesto R_3, ili obrnuto.

Na kraju, treba istaći i to da zbog činjenice da optimalno rješenje *nije jedinstveno*, odgovarajući transportni problem na koji se svodi ovaj problem raspoređivanja može, *osim nađenih optimalnih rješenja*, imati i optimalna rješenja koja *nisu cjelobrojna*. Naravno, takva rješenja *nisu rješenja ovog problema raspoređivanja*. Recimo, nije teško provjeriti da je jedno takvo *necjelobrojno* optimalno rješenje dato sa $x_{1,2} = x_{1,3} = x_{1,5} = x_{2,2} = x_{5,2} = 1/3$, $x_{2,5} = x_{5,3} = 2/3$, $x_{3,1} = x_{4,4} = 1$ uz preostale vrijednosti $x_{i,j}$, $i = 1 \,..\, 5$, $j = 1 \,..\, 5$ jednake nuli. Mada ovo rješenje jasno ne može biti rješenje polaznog problema raspoređivanja u klasičnom smislu, ni ovakvo *necjelobrojno* rješenje nije sasvim lišeno interpretacije. Recimo, moguće je prvog kandidata raspodijeliti trećinu radnog vremena na radno mjesto R_2 i trećinu radnog vremena na radno mjesto R_3 (preostalu trećinu ga ne raspoređujemo *nigdje* jer preostala trećina ide na *nepostojeće* radno mjesto R_5), drugog kandidata raspoređujemo *samo trećinu radnog vremena* na radno mjesto R_2, kandidata R_3 *puno radno vrijeme* na radno mjesto R_3, itd. Ovakvo rješenje očito nije besmisleno ukoliko je dozvoljen *angažman na djelimično radno vrijeme*, a ima prednost nad *klasičnim* rješenjima postavljenog problema raspoređivanja u tome što omogućava da *svi kandidati budu barem djelimično angažirani*.

- ***Primjer***: Riješiti mađarskim algoritmom problem raspoređivanja koji smo ranije rješavali svođenjem na ekvivalentni transportni problem.

Ovaj primjer jasno demonstrira razliku u efikasnosti mađarskog algoritma kao *specijalističkog pristupa* u odnosu na *opće metode* za rješavanje transportnog problema. Nakon dodavanja fiktivnog izvršioca I_4 i množenja funkcije cilja sa –1, dobija se sljedeća tablica:

	M_1	M_2	M_3	M_4
I_1	–3	–2	–5	–4
I_2	–6	–4	–7	–8
I_3	–1	–6	–3	–7
I_4	0	0	0	0

Prvo trebamo obaviti *redukciju ove tablice* (prema *tehnici Flooda*). Izvršićemo redukciju prvo po redovima, a zatim po kolonama (vidjeli smo da može i obrnuto). Nakon redukcije *po redovima,* dolazimo do sljedeće tabele:

	M_1	M_2	M_3	M_4
I_1	2	3	0	1
I_2	2	4	1	0
I_3	6	1	4	0
I_4	0	0	0	0

Sada bi trebalo izvršiti redukciju *po kolonama*. Međutim, kako je najmanji element u svakoj koloni nula, redukcija po kolonama *ne mijenja izgled tablice*, tako da odmah možemo preći na razvrstavanje nula. Nikakve poteškoće se pri tome neće javiti, tako da nakon razvrstavanja dobijamo situaciju kao u sljedećoj tablici, u kojoj imamo *tri nezavisne nule*, tako da nije moguće očitati optimalno rješenje. Dalje, odmah se vidi kako prekriti sve nule u tabeli sa tri linije, pa *nećemo prolaziti* detaljno kroz sve podkorake koraka 4. Uglavnom, nakon obavljenog razvrstavanja i precrtavanja, dolazimo do situacije kao u sljedećoj tablici:

	M_1	M_2	M_3	M_4
~~I_1~~	~~2~~	~~3~~	●	~~1~~
I_2	2	4	1	●
I_3	6	1	4	∅
~~I_4~~	●	∅	∅	∅

Sada odzuimamo najmanji neprectani element (1) od svih neprecrtanih elemenata, a dodajemo ga na dvostruko precrtane elemente. U novoj tabeli se odmah može izvršiti razvrstavanje nula na zavisne i nezavisne (koje ponovo teče bez ikakvih dvosmislica). Po okončanju ovog postupka, dolazimo do sljedeće tablice:

	M_1	M_2	M_3	M_4
I_1	2	3	●	2
I_2	1	3	∅	●
I_3	5	●	3	∅
I_4	●	∅	∅	1

Kako smo dobili skup od četiri nezavisne nule, odmah možemo očitati optimalan raspored $I_1 \rightarrow M_3$, $I_2 \rightarrow M_4$, $I_3 \rightarrow M_2$ i $I_4 \rightarrow M_1$. Vidimo da smo do optimalnog rješenja došli *nakon samo jedne iteracije* (uz malo dodatnih pripremnih radnji), pri čemu je u samoj iteraciji uloženo *veoma malo truda*. Interesantno je napomenuti da smo u ovom primjeru redukciju uradili *drugačijim redoslijedom* (*prvo kolone pa onda redovi*), odmah bismo nakon redukcije i razvrstavanja mogli očitati optimalno rješenje, odnosno *ne bi bila potrebna ni jedna dodatna iteracija nakon tehnike Flooda*. Ovo usput pokazuje da *broj naknadnih iteracija* može ovisiti od redoslijeda kojim je vršena redukcija. Nažalost, *ne može se unaprijed znati koji će redoslijed zahtijevati manje naknadnih iteracija*.

- ***Primjer*** : Naći optimalnan raspored devet diplomata $D_1 - D_9$ u devet ambasada $A_1 - A_9$ sa ciljem da se *minimiziraju ukupni troškovi* pri čemu su troškovi (izraženi u hiljadama EUR) koji slijede ukoliko se neki diplomata rasporedi u neku od ambasada dati u sljedećoj tabeli:

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9
D_1	5	8	13	9	11	4	3	12	7
D_2	8	11	12	7	10	6	5	11	9
D_3	7	5	11	6	9	3	2	8	6
D_4	12	9	8	9	10	5	4	10	8
D_5	3	1	4	2	5	6	8	11	6
D_6	12	6	9	6	10	8	7	13	15
D_7	8	1	4	1	5	7	3	10	11
D_8	14	5	8	5	9	12	10	13	11
D_9	6	3	8	2	7	5	9	8	10

Ovaj primjer je interesantan jer ilustrira situaciju u kojoj pohlepni algoritam *neće dati maksimalan skup nezavisnih nula* u koraku 3, pa će biti potrebne *naknadne korekcije* (inače, ovakve situacije u "ručnoj verziji" algoritma *veoma rijetko nastupaju* u problemima za koje je $n < 10$). Započećemo rješavanje *redukcijom tablice po redovima*, nakon čega dolazimo do sljedeće tablice:

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9
D_1	2	5	10	6	8	1	0	9	4
D_2	3	6	7	2	5	1	0	6	4
D_3	5	3	9	4	7	1	0	6	4
D_4	8	5	4	5	6	1	0	6	4
D_5	2	0	3	1	4	5	7	10	5
D_6	6	0	3	0	4	2	1	7	9
D_7	7	0	3	0	4	6	2	9	10
D_8	9	0	3	0	4	7	5	8	6
D_9	4	1	6	0	5	3	7	6	8

Nastavljamo sa redukcijom *po kolonama*, koja nas dovodi do sljedeće tablice:

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9
D_1	0	5	7	6	4	0	0	3	0
D_2	1	6	4	2	1	0	0	0	0
D_3	3	3	6	4	3	0	0	0	0
D_4	6	5	1	5	2	0	0	0	0
D_5	0	0	0	1	0	4	7	4	1
D_6	4	0	0	0	0	1	1	1	5
D_7	5	0	0	0	0	5	2	3	6
D_8	7	0	0	0	0	6	5	2	2
D_9	2	1	3	0	1	2	7	0	4

Sada je potrebno izvršiti razvrstavanje nula na zavisne i nezavisne. Međutim, u ovom primjeru ćemo imati *mnogo više dvosmislica* nego u dosadašnjim primjerima (tj. situacija u kojima nije jednoznačno određeno koju nulu proglasiti za nezavisnu). Odmah na početku vidimo da niti jedan red niti kolona ne sadrže samo jednu nulu. Potražimo stoga redove ili kolone koji sadrže *dvije nule*. Takav je, recimo, red D_9 (pored kolone A_1), u kojem imamo nule u kolonama A_4 i A_8. Odlučimo se da za nezavisnu proglasimo nulu u koloni A_4, dok ćemo sve ostale nule u redu D_9 i koloni A_4 proglasiti za zavisne. Prelazimo sad na kolonu A_1, koja također sadrži dvije nerazvrstane nule. Odlučimo se da nezavisna nula bude ona u redu D_1 i razvrstajmo sve preostale nule u redu D_1 i koloni A_1 u zavisne. U ovom trenutku nema više niti jedan red niti kolona koji sadrže samo dvije nerazvrstane nule, pa ćemo potražiti redove ili kolone koji sadrže *tri nerazvrstane nule*. Prvi takav red je D_5, koji sadrži nerazvrstane nule u kolonama A_2, A_3 i A_5. Odlučimo se da nezavisna bude recimo nula u koloni A_2 i razvrstajmo preostale nule u redu D_5 i koloni A_2 u zavisne. Sada red D_6 sadrži samo dvije nerazvrstane nule (u kolonama A_3 i A_5). Proglasimo onu u koloni A_3 za nezavisnu, a preostale nule u redu D_6 i koloni A_3 za zavisne. Nakon posljednjeg koraka, pojavljuju se dva reda D_7 i D_8 koji sadrže samo po jednu nerazvrstanu nulu. Proglasimo nulu u redu D_7 i koloni A_5 (jedina

nerazvrstana u redu D_7) za nezavisnu, a sve ostale nule u koloni A_5 za zavisne. Sada više nema redova niti kolona koji sadrže samo jednu ili samo dvije nerazvrstane nule. Prvi red ili kolona koji sadrži samo tri nerazvrstane nule je kolona A_6. Izaberimo da nezavisna bude recimo nula koja se nalazi u redu D_2 i proglasimo sve preostale nule u redu D_2 i koloni A_6 u zavisne. Sada kolona A_7 sadrži samo dvije nerazvrstane nule (u redovima D_3 i D_4). Uzmimo da je nezavisna ona u redu D_3, dok će preostale nule u redu D_3 i koloni A_7 postati zavisne. Nakon ovog koraka, dobijamo kolone A_8 i A_9 koji sadrže samo jednu nerazvrstanu nulu (obje u redu D_4). Odlučimo se da za nezavisnu proglasimo onu koja je u koloni A_8, čime sve preostale nule u redu D_4 postaju zavisne. Sada su sve nule razvrstane, tako da se postupak razvrstavanja završava, i dobijamo situaciju kao u sljedećoj tabeli:

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9
D_1	●	5	7	6	4	∅	∅	3	∅
D_2	1	6	4	2	1	●	∅	∅	∅
D_3	3	3	6	4	3	∅	●	∅	∅
D_4	6	5	1	5	2	∅	∅	●	∅
D_5	∅	●	∅	1	∅	4	7	4	1
D_6	4	∅	●	∅	∅	1	1	1	5
D_7	5	∅	∅	∅	●	5	2	3	6
D_8	7	∅	∅	∅	∅	6	5	2	2
D_9	2	1	3	●	1	2	7	∅	4

Vidimo da smo našli skup od *osam nezavisnih nula*. Međutim, kako je u postupku bilo dosta mjesta na kojima smo vršili *nasumične odluke*, nije posve jasno *da li je ovo i maksimalan skup nezavisnih nula* (vidjećemo uskoro da *nije*).

U svakom slučaju, nastavićemo dalje sa *postupkom precrtavanja* (koji će pokazati da ovaj skup nezavisnih nula *nije maksimalan* i dati sugestije kako ga možemo *povećati*). Pokušaj da *intuitivno pronađemo precrtavanje svih nula u tablici sa samo osam linija neće uspjeti* (što nije čudno, jer takvo precrtavanje *ne postoji*). Stoga pokušajmo obaviti formalni postupak traganja za takvim precrtavanjem. Prvo ćemo označiti jedini red bez nezavisnih nula. To je red D_8. Sada ćemo označiti one kolone koje u redu D_8 imaju zavisne nule, odnosno kolone A_2, A_3, A_4 i A_5. U sljedećem koraku označavamo one redove koji u ovim kolonama imaju nezavisne nule. To su redovi D_5, D_6, D_7 i D_9. Sada označavamo one kolone koji u ovim redovima imaju zavisne nule. To su kolone A_1 i A_8 (kolone A_2, A_3, A_4 i A_5, koje također imaju zavisne nule u ovim redovima, *već su ranije označene*). Nastavljamo sa označavanjem redova koji imaju nezavisne nule u novooznačenim kolonama, a to su redovi D_1 i D_4. Kolone koje nisu ranije bile označene, a koje imaju zavisne nule u ovim redovima su kolone A_6, A_7 i A_9, koje ćemo sada označiti. Postupak se sada na osnovu podkoraka 4.3. *prekida*, jer smo upravo označili *kolonu* A_9 *u kojoj nema nezavisnih nula*. Na kraju postupka, dobijamo situaciju kao u sljedećoj tabeli (kraj strelica su, radi lakšeg snalaženja, prikazani redni brojevi koji označavaju u kojoj "turi" označavanja je označen odgovarajući red odnosno kolona):

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	
D_1	●	5	7	6	4	∅	∅	3	∅	← 5
D_2	1	6	4	2	1	●	∅	∅	∅	
D_3	3	3	6	4	3	∅	●	∅	∅	
D_4	6	5	1	5	2	∅	∅	●	∅	← 5
D_5	∅	●	∅	1	∅	4	7	4	1	← 3
D_6	4	∅	●	∅	∅	1	1	1	5	← 3
D_7	5	∅	∅	∅	●	5	2	3	6	← 3
D_8	7	∅	∅	∅	∅	6	5	2	2	← 1
D_9	2	1	3	●	1	2	7	∅	4	← 3
	↑	↑	↑	↑	↑	↑	↑	↑	↑	
	4	2	2	2	2	6	6	4	6	

Činjenica da je postupak označavanja *prekinut prije vremena* ukazuje na to da je *uzaludno* tražiti precrtavanje svih nula sa osam linija. Zaista, ukoliko bismo dalje nastavili postupak označavanja *ignorirajući činjenicu da trebamo odustati*, označili bismo redove D_1 i D_2, i nakon toga bismo svakako morali završiti postupak, jer više ne bi bilo neoznačenih kolona koje imaju zavisne nule u novooznačenim redovima. Međutim, nakon toga bi bili označeni *svi redove i sve kolone*, tako da bismo trebali precrtati *sve kolone* (i ni jedan red). Međutim, to je precrtavanje sa *devet* linija, a ne sa osam, koliko smo pokušali da nađemo.

Ovo znači da je *pronađen put* koji ide naizmjenično horizontalno i vertikalno i koji spaja naizmjenično zavisne i nezavisne nule, a koji počinje od neke zavisne nule u *koloni u kojoj smo prekinuli postupak* (tj. A_9) i završava u nekoj zavisnoj nuli u *nekom od redova od kojeg smo započeli postupak*. Takav put lako nalazimo "razmotavajući" postupak označavanja *unazad*. Kolonu A_9 smo označili prilikom razmatranja redova D_1 i D_4. Možemo uzeti *bilo koji od njih*, recimo D_1 (činjenica da *ovaj izbor nije jedinstven* ukazuje na to i da *traženi put nije jedinstven*). Dakle, put počinje od zavisne nule u redu D_1 i koloni A_9. Dalje, red D_1 smo označili kada smo razmatrali kolonu A_1. Stoga je sljedeća "stanica" na traženom putu nezavisna nula u presjeku reda D_1 i kolone A_1. Kolonu A_1 smo označili prilikom razmatranja reda D_5, pa put dalje vodi preko zavisne nule u presjeku reda D_5 i kolone A_1. Red D_5 smo označili kada smo razmatrali kolonu A_2, tako da je sljedeća tačka u traženom putu nezavisna nula u redu D_5 i koloni A_2. Konačno, kolonu A_2 smo označili kada smo razmatrali red D_8, od kojeg smo i započeli postupak. Slijedi da se traženi povećavajući put završava u zavisnoj nuli u presjeku reda D_8 i kolone A_2 (pored ovog, postoji još jedan povećavajući put koji bismo mogli naći da smo uzeli red D_4 umjesto reda D_1). Nađeni povećavajući put je prikazan na sljedećoj slici:

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9
D_1	●	5	7	6	4	∅	∅	3	∅
D_2	1	6	4	2	1	●	∅	∅	∅
D_3	3	3	6	4	3	∅	●	∅	∅
D_4	6	5	1	5	2	∅	∅	●	∅
D_5	∅	●	∅	1	∅	4	7	4	1
D_6	4	∅	●	∅	∅	1	1	1	5
D_7	5	∅	∅	∅	●	5	2	3	6
D_8	7	∅	∅	∅	∅	6	5	2	2
D_9	2	1	3	●	1	2	7	∅	4

Ukoliko sada u svim "tačkama skretanja" nađenog povećavajućeg puta promijenimo "status" nula (tj. one koje su nezavisne proglasimo za zavisne, a one koje su zavisne za nezavisne), dobićemo *novi skup nezavisnih nula* koji će imati jednu *nezavisnu nulu više*. Konkretno, nule na pozicijama $(1,9)$, $(5,1)$ i $(8,2)$ postaće *nove nezavisne nule*, dok će nule na pozicijama $(1,1)$ i $(5,2)$ postati *zavisne*. Novodobijena situacija prikazana je u sljedećoj tablici:

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9
D_1	∅	5	7	6	4	∅	∅	3	●
D_2	1	6	4	2	1	●	∅	∅	∅
D_3	3	3	6	4	3	∅	●	∅	∅
D_4	6	5	1	5	2	∅	∅	●	∅
D_5	●	∅	∅	1	∅	4	7	4	1
D_6	4	∅	●	∅	∅	1	1	1	5
D_7	5	∅	∅	∅	●	5	2	3	6
D_8	7	●	∅	∅	∅	6	5	2	2
D_9	2	1	3	●	1	2	7	∅	4

Kako smo dobili skup nezavisnih nula koji sadrži *devet nezavisnih nula*, optimalno rješenje je nađeno i nema potrebe da idemo dalje. Dakle, optimalni raspored diplomata po ambasadama glasi $D_1 \to A_9$, $D_2 \to A_6$, $D_3 \to A_7$, $D_4 \to A_8$, $D_5 \to A_1$, $D_6 \to A_3$, $D_7 \to A_5$, $D_8 \to A_2$ i $D_9 \to A_4$. Troškovi ovakvog aranžmana iznose $Z = 7 + 6 + 2 + 10 + 3 + 9 + 5 + 5 + 2 = 49$ hiljada EUR. Treba još istaći da nađeno optimalno rješenje *nije jedinstveno*. Na primjer, da smo odabrali da povećanje skupa nezavisnih nula obavimo koristeći *drugi povećavajući put*, došli bismo do drugačijeg optimalnog rješenja.

Detaljniji uvid u mađarski algoritam

Mađarski algoritam, u formi kakva je prethodno prikazana, djeluje *prilično misteriozno* i na prvi pogled je nejasno *zbog čega taj algoritam uopće radi* i *kakva mu je efikasnost*. Stoga ćemo prvo dati neka *intuitivna objašnjenja* prirode algoritma, a nakon toga ćemo pokazati da je mađarski algoritam u osnovi *dualni metod*, što će dati i neke ideje kako se algoritam može učiniti *efikasnijim*.

Prvo ćemo razmotriti smisao redukcija koje se izvode u koraku 1 i 2. Ono što treba primijetiti je da se optimalno rješenje *neće promijeniti* ukoliko se u matrici **C** elementi ma kojeg reda ili ma koje kolone povećaju ili umanje za isti iznos (jedino će se promijeniti *optimalna vrijednost funkcije cilja*). Ovu činjenicu nije teško dokazati (kasnije ćemo se u to uvjeriti razmatrajući dualni problem problema raspoređivanja), a prilično se uklapa i u intuitivno rezonovanje. Dakle, koraci 1 i 2 daju novu matricu umjesto matrice **C** kojoj odgovara isto optimalno rješenje kao i za polazni problem. Međutim, očigledno je da će nakon koraka 1 transformirana matrica imati sve elemente *nenegativne* i da će pri tome *barem jedan element u svakom redu biti jednak nuli*. Isto tako, nakon koraka 2 će i *u svakoj koloni barem jedan element biti jednak nuli*. Dakle, nakon prva dva koraka dobijamo matricu koja u svakom redu i svakoj koloni ima barem jednu nulu.

Razmotrimo sada smisao koraka 3. Ukoliko u transformiranoj matrici uspijemo odabrati takav skup nula da *u svakom redu i svakoj koloni bude tačno jedna odabrana nula*, jasno je da tada rješenje u kojem je $x_{i,j} = 1$ samo na pozicijama gdje su odabrane nule *mora biti optimalno*, s obzirom da se traži *minimum* funkcije cilja. Prvo, jasno je da je takvo rješenje *dopustivo* (imamo tačno jednu nenultu vrijednost za $x_{i,j}$ u svakom redu i koloni). Drugo, vrijednost *transformirane* funkcije cilja (tj. funkcije cilja dobijene na osnovu transformirane matrice) za tako formirano rješenje očigledno je jednaka nuli, dok će sva druga dozvoljena rješenja davati vrijednost transformirane funkcije cilja koje su *veće ili jednake nuli* (zbog *nenegativnosti* koeficijenata transformirane matrice). Međutim, takav skup odabranih nula nije ništa drugo nego *maksimalni skup nezavisnih nula veličine n*. S druge strane, ukoliko maksimalan skup nezavisnih nula *ima manje od n elemenata*, na osnovu njega *nije moguće formirati dopustivo rješenje* (jer svako dopustivo rješenje mora imati tačno *n* nenultih promjenljivih). Ovim je razjašnjen smisao koraka 3 (i koraka 7 na osnovu kojeg se određuje optimalno rješenje kada korak 3 uspije dati skup nezavisnih nula veličine *n*). Drugim riječima, objašnjene su ideje koje stoje iza *tehnike Flooda*.

Ostavimo za trenutak korak 4 po strani. Što se tiče koraka 5 i 6, korak 6 se isto svodi na povećanje ili smanjenje svih elemenata nekih redova odnosno kolona za *isti iznos* Δ, tako da operacije koje se provode u koraku 6 kao rezultat ponovo daju novu matricu kojoj odgovara *isto optimalno rješenje*. Pored toga, zbog načina kako je odabran Δ u koraku 5, jasno je i da će novodobijena matrica ponovo imati sve nenegativne koeficijente. Međutim, treba primijetiti da će novodobijena matrica *imati nule i na nekim mjestima gdje ih ranije nije bilo*. Tačnije, nove nule će biti na mjestima *svih neprecrtanih elemenata koji su jednaki* Δ (iz načina kako je Δ određen, postoji *barem jedno takvo mjesto*). S druge strane, može se desiti da *neke nule nestanu*. Konkretno, ukoliko je postojala nula koja je precrtana i horizontalnom i vertikalnom linijom, takva nula će *nestati*, jer se *sabira sa* Δ. Stoga je pitanje *da li nakon koraka* 6 *pravimo ikakav progres*. Razmotrimo stoga šta se tačno dešava u koraku 6. Neka smo nakon koraka 4 precrtali p redova i q kolona (ranije pomenuti *Kőnig- Egerváryjev teorem* tvrdi da je $p + q = k$, gdje je k veličina maksimalnog skupa nezavisnih nula). Tada ćemo veličinu Δ oduzeti od ukupno $(n-p)(n-q)$ elemenata, a dodati na $p\,q$ elemenata. Slijedi da će se suma svih elemenata u matrici umanjiti za iznos $(n-p)(n-q)\Delta - p\,q\,\Delta = n(n-p-q)\Delta = n(n-k)\Delta$. Kako je $k < n$, ovaj iznos je *striktno pozitivan*, što znači da se, nakon svakog provođenja koraka 6, suma svih elemenata u matrici *striktno smanjuje*. S druge strane, elementi matrice nikada ne mogu postati negativni, tako da sve ovo daje nadu da će se, sa vremenom, broj nula u matrici postepeno *povećavati*, što povećava šansu da se u koraku 3 uspije pronaći skup nezavisnih nula veličine *n*.

Još jedna stvar koju treba primijetiti je da provođenje koraka 6 nikada *ne može smanjiti* veličinu maksimalnog skupa nezavisnih nula. Zaista, ako pogledamo kako se provodi korak 4, vidjećemo da se ne može desiti da neka nezavisna nula bude precrtana i horizontalnom i vertikalnom nulom. Slijedi da će, nakon koraka 6, svaka nula koja je bila nezavisna sigurno ostati nula. Drugim riječima, isti skup nezavisnih nula koji smo imali prije koraka 6 *i dalje može ostati skup nezavisnih nula*, samo što nakon koraka 6 *možda može postojati i neki drugi, veći skup nezavisnih nula*. Detaljnija analiza zasnovana na nekim stavovima iz teorije grafova pokazuje da se nakon *najviše n iteracija* (a obično i nakon *mnogo manje*) veličina maksimalnog skupa nezavisnih nula *mora povećati za jedinicu*. Ukoliko početna veličina maksimalnog skupa nezavisnih nula dobijena nakon prvog izvršavanja koraka 3 iznosi k, slijedi da se optimalno rješenje mora postići u *najviše* $n(n-k)$ *iteracija* (a obično mnogo manje), jer se veličina skupa nezavisnih nula mora povećati za $n-k$, a svako povećanje za 1 se postiže u najviše n iteracija. Slijedi da je maksimalan broj iteracija ograničen odozgo sa n^2. Kako je broj operacija u svakoj iteraciji približno proporcionalan sa n^2, slijedi da broj ukupnih elementatnih operacija koje se izvršavaju tokom mađarskog algoritma ni u najgorem slučaju ne prelazi iznos proporcionalan sa n^4. Ovo je dobar rezultat, s obzirom da je količina ulaznih podataka n^2 (tako da je ovaj algoritam *kvadratne složenosti* u odnosu na količinu ulaznih podataka). Međutim, uskoro ćemo vidjeti da se vrijeme izvršavanja može, uz neke trikove, *svesti na iznos približno proporcionalan sa* n^3 (uz upotrebu nekih komplikovanih struktura podataka i vrlo komplikovanu implementaciju, može se postići i još bolje vrijeme izvršavanja reda $n^2 \log_2 n$).

Razmotrimo sada kako se mađarski algoritam *može interpretirati kao dualni metod*. S obzirom da se problem raspoređivanja može posmatrati kao *specijalan slučaj transportnog problema*, njegov dualni problem glasi

$$\arg\max W = \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$$

p.o.

$$u_i + v_j \le c_{i,j}, \quad i = 1..n, \quad j = 1..n$$

Uvođenjem izravnavajućih (dopunskih) promjenljivih $d_{i,j}$, $i = 1..n$, $j = 1..n$ ovaj problem se može napisati u sljedećem obliku:

$$\arg\max W = \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j$$

p.o.

$$u_i + v_j + d_{i,j} = c_{i,j}, \quad i = 1..n, \quad j = 1..n$$

$$d_{i,j} \ge 0$$

Naravno, odavde direktno slijedi da je $d_{i,j} = c_{i,j} - u_i - v_j$, $i = 1..n$, $j = 1..n$. Također, vidimo da se lako može naći *neko dopustivo rješenje* ovog dualnog problema (dovoljno je recimo uzeti *da su sve promjenljive* u_i, $i = 1..n$ i v_j, $j = 1..n$ *jednake nuli*). Međutim, znamo da u optimalnom rješenju primalnog problema imamo tačno n vrijednosti $x_{i,j}$ koje su različite od nule. Na osnovu *svojstva oslabljene komplementarnosti* prema kojem je $x_{i,j} d_{i,j} = 0$ za sve $i = 1..n$ i $j = 1..n$, slijedi da u optimalnom rješenju barem n izravnavajućih dualnih promjenljivih $d_{i,j}$ moraju biti jednake nuli, odnosno barem n ograničenja duala tipa $u_i + v_j \le c_{i,j}$ moraju biti zadovoljena *do na jednakost*. Stoga bi dobra ideja bila potražiti *početno dopustivo dualno rješenje kod kojeg će ovo biti ispunjeno*, u nadi da, ako budemo imali sreće, *možda odmah ono bude optimalno*.

Jedan logičan izbor za početno dualno rješenje koje zadovoljava ovaj uvjet je da stavimo da je u_i jednako najmanjem elementu u i-tom redu matrice **C**, za sve $i = 1..n$, i da stavimo da je $v_j = 0$, $j = 1..n$. Kako je tada $d_{i,j} = c_{i,j} - u_i - v_j = c_{i,j} - u_i$, slijedi da se matrica izravnavajućih dualnih promjenljivih **D** u ovom slučaju dobija *oduzimanjem od svakog reda matrice* **C** *najmanjih elemenata tih redova*, a to je upravo korak 1 mađarskog algoritma. Još pogodnije dualno rješenje (sa više izravnavajućih dualnih promjenljivih jednakih nuli) dobije se ukoliko se ne uzme da su sve promjenljive v_j jednake nuli, nego se, nakon što se promjenljive u_i odrede na gore opisani način, uzme da je v_j za neko konkretno j jednako najmanjoj od vrijednosti $c_{i,j} - u_i$ za sve $i = 1..n$, i tako redom za sve j od 1 do n. Drugim riječima, uzimamo

$$u_i = \min_{j=1..n} c_{i,j}, \quad i = 1..n$$

$$v_j = \min_{i=1..n} (c_{i,j} - u_i), \quad j = 1..n$$

Nakon ovakvog izbora osnovnih dualnih promjenljivih, nije teško vidjeti da je matrica izravnavajućih dualnih promjenljivih **D** upravo *matrica koja se dobija nakon prva dva koraka mađarskog algoritma*. Zapravo, tokom cijelog mađarskog algoritma, matrice sa kojima radimo (reducirane matrice) uvijek su matrice izravnavajućih dualnih promjenljivih koje odgovaraju dualno dopustivom baznom rješenju u kojem se trenutno nalazimo. Vrijedi još napomenuti da u slučaju da se prvo vrši redukcija matrice po kolonama pa onda po redovima, takvoj odluci odgovara sljedeći izbor početnih vrijednosti za dualne promjenljive:

$$v_j = \min_{i=1..n} c_{i,j}, \quad j = 1..n$$

$$u_i = \min_{j=1..n} (c_{i,j} - v_j), \quad i = 1..n$$

Krenuli smo, dakle, od jednog dualno dopustivog rješenja (za koje se može pokazati i da je *bazno*). Znamo da njemu komplementarno rješenje primala može imati nenulte vrijednosti za $x_{i,j}$ samo tamo gdje je $d_{i,j} = 0$. Kako sve nenulte vrijednosti $x_{i,j}$ mogu biti samo jedinice, pokušavamo da pronađemo da li je moguće postaviti $x_{i,j} = 1$ na neka mjesta gdje je $d_{i,j} = 0$, a da pri tome primalno rješenje *bude dopustivo*, tj. da imamo *tačno jednu jedinicu* u svakom redu i svakoj koloni. Zahtjev za *najviše jednom jedinicom* u svakom

redu i koloni svodi se na traženje *skupa nezavisnih nula* u matrici **D**. Ukoliko uspijemo postići da tih nezavisnih nula ima *n*, dobićemo tačno jednu jedinicu u svakom redu i koloni. Tako će dobijeno rješenje biti i *primalno dopustivo*, a prema općoj teoriji dualnosti, ono onda mora biti i *optimalno*. S druge strane, ukoliko ne možemo dobiti *n* nezavisnih nula, nećemo moći dobiti primalno dopustivo rješenje (u makar nekom redu i koloni nećemo moći formirati zbir 1). Slijedi da je onda tekuće rješenje *superoptimalno* (daje funkciji cilja vrijednost *bolju od optimalne*, ali *nije dopustivo*). Ovo je smisao koraka 3, pri čemu se samo traganje za maksimalnim skupom u matrici **D** vrši interpretacijom ovog traganja kao *problema iz teorije grafova* (problem *maksimalnog uparivanja*) i korištenjem *od ranije poznatih tehnika* za rješavanje ovog problema.

Pogledajmo sada kako se koraci 5 i 6 mogu interpretirati kao *prelazak sa jednog dopustivog na drugo dopustivo dualno rješenje*. U koraku 6 se od nekih *redova* matrice **D** *oduzima* vrijednost Δ određena u koraku 5, dok se na neke *kolone* matrice **D** ista vrijednost *dodaje*. Međutim, kako je $d_{i,j} = c_{i,j} - u_i - v_j$, oduzimanje Δ od nekog *i*-tog reda matrice **D** svodi se na *povećanje* promjenljive u_i za Δ, dok se dodavanje Δ na neku *j*-tu kolonu matrice **D** svodi na *umanjenje* promjenljive v_j za Δ. Dakle, korak 6 je *transformacija vrijednosti dualnih promjenljivih*, odnosno prelazak na *novo dualno dopustivo rješenje*. Vrijednost Δ je odabrana tako da promjena bude *maksimalno moguća* a da novo rješenje bude i dalje *dualno dopustivo*. Ovo samim tim garantira da će novo rješenje *također ostati bazno rješenje*. Treba još vidjeti *šta se pri tome dešava sa funkcijom cilja duala*. Sumiramo li obje strane jednakosti $d_{i,j} = c_{i,j} - u_i - v_j$ po *i* i *j* dobijamo:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{i,j} = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} - \sum_{i=1}^{n}\sum_{j=1}^{n} u_i - \sum_{i=1}^{n}\sum_{j=1}^{n} v_j = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} - n\sum_{i=1}^{n} u_i - n\sum_{j=1}^{n} v_j$$

Odavde je:

$$W = \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} - \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{i,j}$$

Prvi član u izrazu sa desne strane ostaje uvijek isti iz iteracije u iteraciju, jer se matrica **C** ne mijenja. Međutim, vidjeli smo da se, zbog načina kako je izvršen korak 4, suma svih elemenata u matrici **D** nakon koraka 6 *smanjuje* za iznos $n(n-k)\Delta$. Odavde slijedi da se u svakoj iteraciji, vrijednost funkcije cilja *W* *povećava* za iznos $(n-k)\Delta$. Drugim riječima, prelazimo iz jednog dualno dopustivog baznog rješenja u *bolje* (sa aspekta maksimizacije *W*) dualno dopustivo bazno rješenje, pri čemu se putokaz za taj prelazak dobija ponovo rješavanjem jednog *pomoćnog problema iz teorije grafova* (problem *minimalnog pokrivača vrhova*) koji se provodi u koraku 4. Konačno zaključujemo da mađarski algoritam u svakoj iteraciji vrši prelazak iz jednog dualno dopustivog u drugo *bolje* dualno dopustivo rješenje. Iteracije se vrše sve dok se ne postigne i *primalna dopustivost*, a samim tim i *optimalnost* (kako dualnog, tako i primalnog rješenja).

Treba još istaći da se u opisima grafovske verzije mađarskog algoritma, osnovne dualne promjenljive *eksplicitno spominju*, samo *ne pod tim imenom*. Naime, one se pridružuju *čvorovima grafa* i nazivaju ***potencijali*** ili ***oznake*** (engl. *labels*) ***čvorova***.

Prilagođavanje mađarskog algoritma za računarsku implementaciju

Mađarski algoritam, u formi kako je prethodno izložena, prilagođen je za *ručno rješavanje* i stoga nije osobito pogodan za *implementaciju na računaru* (inače, verzije mnogih algoritama prilagođene za ručni rad preferiraju *manji broj iteracija*, a *više posla u svakoj iteraciji*, dok je kod verzija prilagođenih za računasku izvedbu *obrnuto*). Pri tome je velika pažnja posvećena tome da se *što je god moguće više smanji potreba za izvođenjem podkoraka* 4.5., s obzirom da je njegova izvedba donekle *mukotrpna* pri ručnom radu, jer predstavlja izvjesnu vrstu "popravke" nečega što "nije prethodno urađeno na najbolji mogući način". Međutim, na računaru izvedba ovog podkoraka nije nikakav osobit problem, pa ne smeta čak i ako se ovaj korak izvodi *često*. Slijedi da se nakon koraka 6 *ne moramo vraćati nazad na korak* 3, već je *dovoljno vratiti se na korak* 4. Zaista, maksimalni skup nezavisnih nula u matrici **D** kakav je bio prije koraka 6 ostaje skup nezavisnih nula i nakon koraka 6, samo što nakon njega *možda neće više biti maksimalan*. Ukoliko on ostane maksimalan, dobro jeste, a ukoliko *ne ostane*, to će u koraku 4 biti primijećeno, te ćemo izvršiti njegovu *korekciju* (u podkoraku 4.5) tako da ćemo dobiti novi maksimalni skup nezavisnih nula.

Na ovaj način, korak 3 se obavlja *samo prvi put*, a *ne u svakoj iteraciji*. Ovo je *velika ušteda u vremenu*, jer je korak 3 *vremenski prilično zahtjevan* (traži vrijeme proporcionalno sa n^2 ako koristimo pohlepni postupak, ili još više ukoliko bismo koristili neki postupak koji *garantira* nalaženje *maksimalnog* skupa nezavisnih nula, ali to se definitivno *ne isplati*). Ovako, u svakoj iteraciji se prosto pretpostavlja da je

maksimalni skup nezavisnih nula *ostao isti*, a ukoliko se pokaže da to nije tako, vrši se *korekcija*, koja se može obaviti u vremenu proporcionalnom sa n. Iz prethodne diskusije je jasno da *ukupan* broj korekcija tokom izvođenja algoritma neće preći $n-k$, tako da je ovo definitivno isplativa modifikacija sa aspekta vremena izvršavanja.

Sljedeća stvar koju treba uočiti sa aspekta poboljšanja efikasnosti implementacije je da nam nenulti elementi matrice **D** nisu potrebni *niti u jednom koraku algoritma* osim koraka 5. Jedino što nam je potrebno je da znamo koji su elementi u matrici **D** jednaki nuli. Međutim, znamo da je uvjet $d_{i,j} = 0$ ekvivalentan sa uvjetom $u_i + v_j = c_{i,j}$. Slijedi da je dovoljno umjesto matrice **D** čuvati *samo vrijednosti osnovnih dualnih promjenljivih* u_i, $i = 1 \,..\, n$, i v_j, $j = 1 \,..\, n$, jer je računaru praktično *svejedno* da li testira uvjet $u_i + v_j = c_{i,j}$ ili uvjet $d_{i,j} = 0$. S druge strane, čovjek *vizualno* mnogo lakše uočava nule u tablici nego da li je neko polje jednako zbiru neke dvije veličine, tako da je pri ručnom radu eksplicitni prikaz matrice **D** itekako koristan. Dalje, pogledajmo korak 6 u kojem se neki elementi matrice **D** uvećavaju, a neki umanjuju za Δ. Vidjeli smo prilikom analize da se mijenja ukupno $(n-p)(n-q)+p\,q$ elemenata, što je za tipične vrijednosti p i q reda n^2 (npr. za $p = q = n/2$ ovo je tačno $n^2/2$ elemenata). S druge strane, vidjeli smo da je to *potpuno ekvivalentno* uvećavanju ukupno p promjenljivih u_i i umanjivanju ukupno q promjenljivih v_j za Δ, odnosno *izmjeni* ukupno $p+q=k$ promjenljivih (dakle, *manje od* n, jer je $k < n$). Iz ovoga slijedi da elemente matrice **D** *uopće nije isplatno računati i ažurirati*, nego je potrebno samo ažurirati vrijednosti osnovnih dualnih promjenljivih, kojih ima znatno manje nego elemenata matrice **D**. Sve što nam je potrebno može se odrediti iz matrice **C** i osnovnih dualnih promjenljivih. Doduše, za računanje Δ potrebno je naći najmanji od ukupno $(n-p)(n-q)$ elemenata $d_{i,j}$ u matrici **D**. Ako matrice **D** nemamo, umjesto toga tražimo najmanju od ukupno $(n-p)(n-q)$ veličina $c_{i,j}-u_i-v_j$, što *za računar nije praktično nikakva razlika* (ali jeste za čovjeka). Ipak, ovo je još uvijek mnogo računanja, koje ne dozvoljava da ukupno vrijeme izvršavanja padne ispod iznosa proporcionalnog sa n^4, ali ćemo uskoro vidjeti i kako se ovo može poboljšati.

Razmotrimo sada šta se može učiniti u koraku 4. Jasno je da je precrtavanje koje se vrši u podkoraku 4.6 samo tehnika koja pri ručnom radu pomaže da *lakše uočimo* među kojim elementima matrice treba tražiti vrijednosti Δ i koje elemente treba korigovati. Ovaj podkorak *nema nikakvog smisla pri računarskoj implementaciji*, jer sve što nam treba možemo saznati iz informacije *koji su redovi i koje kolone označeni*. Dalje, implementaciju otežavaju neke konstrukcije u pojedinim podkoracima koraka 4 koje su *u množini*, poput "označimo redove" (u podkoracima 4.1 i 4.4), "označimo kolone" (u podkoraku 4.2), itd. Ključno je primijetiti da će algoritam raditi i ukoliko ove konstrukcije zamijenimo konstrukcijama *u jednini* poput "označimo (neki) red", itd. Doduše, na ovaj način se neće označiti svi redovi i kolone koji bi trebali, te se u koraku 5 *neće izmijeniti svi elementi matrice* **D** *koji bi trebali* (tačnije, sve *dualne promjenljive koje bi trebale*, jer mi zapravo i ne ažuriramo matricu **D**). Međutim, na osnovu ranije provedene analize, jasno je da će i tako dobijeno novo rješenje biti *dualno dopustivo* i *bolje od prethodnog*. Ne trebamo se bojati da ćemo time nešto bitno propustiti, jer će propuštene korekcije *biti nadoknađene kasnije*. Doduše to će *povećati broj neophodnih iteracija* (što pri ručnom radu nije poželjno svojstvo), ali će *smanjiti količinu posla* u pojedinim iteracijama, tako da se pokazuje da *ukupna količina posla ostaje ista*, ali uz *jednostavniju implementaciju*. Također treba primijetiti da označavanje redova i kolona *ne treba svaki put vršiti ispočetka*, nego *samo kada dođe do promjene rasporeda nezavisnih nula* (tj. kada izvršimo podkorak 4.5). Dakle, kada god pravimo neku izmjenu koja ne dovodi do promjene rasporeda nezavsnih nula, postojeće označavanje redova i kolona i dalje *ostaje ispravno*, tako da samo treba *nastaviti dalje* sa obilježavanjem, *počev od trenutno zatečenog označavanja*. Označavanje će *biti moguće nastaviti*, jer će provedene izmjene dualnih promjenljivih dovesti do pojave nula u matrici **D** tamo *gdje ih ranije nije bilo*.

Na osnovu svega do sada rečenog, možemo skicirati tok mađarskog algoritma pogodan za računarsku implementaciju:

1. Odrede se neke dopustive vrijednosti za promjenljive u_i, $i = 1 \,..\, n$. Obično se uzima da je u_i jednako najmanjem elementu u i-tom redu matrice **C**.
2. Odrede se neke dopustive vrijednosti za promjenljive v_j, $j = 1 \,..\, n$. Obično se uzima da je v_j jednako najmanjoj od vrijednosti $c_{i,j} - u_i$ za sve $i = 1 \,..\, n$. U nekim izvedbama se uzima da su sve promjenljive v_j na početku jednake nuli, ali na taj način se *povećava broj iteracija*.
3. Nekim postupkom (recimo opisanim pohlepnim postupkom) pronađe se *neki skup nezavisnih nula* u *matrici izravnavajućih dualnih promjenljivih* **D** (što veći, to bolji, ali se *ne mora* insistirati da bude maksimalan). Pri tome se elementi matrice **D** *ne računaju*, nego se koristi činjenica da se u matrici **D** nula na poziciji (i,j) nalazi *ako i samo ako vrijedi* $u_i + v_j = c_{i,j}$.

4.1. *Označimo neki red koji ne sadrži nezavisne nule*. Ukoliko takvih redova *nema*, prelazimo na korak 7.

4.2. *Označimo neku kolonu* koja u *označenim redovima ima zavisnu nulu*, a koja *ranije nije bila označena*. Ukoliko takvih kolona *nema*, prelazimo na korak 5.
4.3. Ukoliko upravo označena kolona *ne sadrži niti jednu nezavisnu nulu*, prelazimo na korak 4.5.
4.4. Označimo *red* koji ima *nezavisnu nulu* u *upravo označenoj koloni* i vraćamo se na korak 4.2.
4.5. Sada treba pronaći (razmotavanjem "unazad" prateći redoslijed kojim su označavane kolone i redovi) *povećavajući put* koji ide *naizmjenično horizontalno i vertikalno* i *naizmjenično posjećuje zavisne i nezavisne nule*, a počinje u *koloni koju smo upravo označili* i završava u *prvom označenom redu*. Duž takvog puta *svaku zavisnu nulu proglasimo za nezavisnu a svaku nezavisnu nulu za zavisnu*, čime dobijamo novi *veći* skup nezavisnih nula. Nakon toga, poništimo *sva označavanja* i vraćamo se na korak 4.1.
5. Pronađemo *najmanji element* Δ među vrijednostima $c_{i,j} - u_i - v_j$ (to jest, među vrijednostima izravnavajućih dualnih promjenljivih $d_{i,j}$) za one indekse *i* koji odgovaraju *označenim redovima* i one indekse *j* koji odgovaraju *neoznačenim kolonama*.
6. *Uvećamo* za Δ dualne promjenljive u_i za one indekse *i* koji odgovaraju *označenim redovima*, a *umanjimo* za Δ dualne promjenljive v_j za one indekse *j* koji odgovaraju *označenim kolonama*, nakon čega se vraćamo na korak 4.2.
7. Optimalno rješenje je *nađeno* i očitava se na sljedeći način. Za sve $i = 1..n$ i $j = 1..n$ ukoliko se na poziciji (i, j) u matrici izravnavajućih dualnih promjenljivih nalazi *nezavisna nula*, tada je $x_{i,j} = 1$, inače je $x_{i,j} = 0$.

Ovdje smo numeraciju koraka izvršili tako da se lakše prati paralela sa prvobitnom "ručnom verzijom". Ovako modificiran algoritam je *znatno ubrzan* (pri izvedbi na računaru) u odnosu na prvobitnu verziju. Međutim, u njemu još uvijek ima pojedinih "uskih grla" koji ne dozvoljavaju da vrijeme izvršavanja ovog algoritma bude reda n^3 (prvenstveno korak 5, a probleme pravi i množina "označenim redovima" u koraku 4.2). Prije nego što vidimo šta se može učiniti po ovom pitanju, demonstriraćemo tok ovako modificiranog algoritma na jednom primjeru.

- ***Primjer***: Riješiti ponovo problem raspoređivanja pet kandidata $K_1 - K_5$ na četiri radna mjesta $R_1 - R_4$ koji je već ranije rješavan verzijom mađarskom algoritma prilagođenom za *ručni rad*, ali ovaj put gore opisanom verzijom mađarskog algoritma prilagođenom za *izvođenje na računaru*.

Krenućemo, kao i u "ručnoj verziji" algoritma, od polazne matrice cijena za ekvivalentni balansirani problem minimizacije:

	R_1	R_2	R_3	R_4	R_5
K_1	–5	–6	–5	–1	0
K_2	–4	–6	–4	–1	0
K_3	–8	–6	–7	–6	0
K_4	–2	–4	–4	–4	0
K_5	–6	–10	–9	–4	0

U prva dva koraka, potrebno je odrediti početne vrijednosti dualnih promjenljivih. Da bismo mogli lakše napraviti paralelu u odnosu na "ručnu verziju" algoritma, prvo ćemo odrediti dualne promjenljive v_j traženjem najmanje vrijednosti u svakoj koloni, a zatim dualne promjenljive u_i traženjem najmanje vrijednosti među vrijednostima $c_{i,j} - v_j$ za sve redove, s obzirom da smo kod "ručne verzije" vršili redukciju matrice prvo po kolonama pa onda po redovima. Tako dobijamo:

$v_1 = \min\{-5, -4, -8, -2, -6\} = -8$
$v_2 = \min\{-6, -6, -6, -4, -10\} = -10$
$v_3 = \min\{-5, -4, -7, -4, -9\} = -9$
$v_4 = \min\{-1, -1, -6, -4, -4\} = -6$
$v_5 = \min\{0, 0, 0, 0, 0\} = 0$
$u_1 = \min\{-5-(-8), -6-(-10), -5-(-9), -1-(-6), 0-0\} = 0$
$u_2 = \min\{-4-(-8), -6-(-10), -4-(-9), -1-(-6), 0-0\} = 0$
$u_3 = \min\{-8-(-8), -6-(-10), -7-(-9), -6-(-6), 0-0\} = 0$
$u_4 = \min\{-2-(-8), -4-(-10), -4-(-9), -4-(-6), 0-0\} = 0$
$u_5 = \min\{-6-(-8), -10-(-10), -9-(-9), -4-(-6), 0-0\} = 0$

Nakon što se odrede osnovne dualne promjenljive, njima odgovara sljedeća tablica *izravnavajućih* dualnih promjenljivih $d_{i,j}$ (kao što je već rečeno, ova tablica se *uopće ne izračunava* pri računarskoj implementaciji, ali je navodimo radi preglednijeg opisa koraka koji slijede):

	R_1	R_2	R_3	R_4	R_5
K_1	3	4	4	5	0
K_2	4	4	5	5	0
K_3	0	4	2	0	0
K_4	6	6	5	2	0
K_5	2	0	0	2	0

Jasno je da je ovo *ista tablica* kakva se dobija nakon prva koraka "ručne verzije" algoritma. U nastavku će se, radi lakšeg praćenja, u svakom koraku prikazivati upravo te tablice koje se *ne računaju*, jer je na njima lakše pratiti šta se dešava. Sada je, kao i kod "ručne verzije" algoritma potrebno razvrstati sve nule u ovoj tablici na zavisne i nezavisne (kako ova tablica uopće nije ni računata, lokacije gdje se u njoj nalaze nule obavlja se testiranjem uvjeta $u_i + v_j = c_{i,j}$). Razvrstavanje obavljamo istim postupkom kao i kod "ručne verzije", tako da dolazimo do sljedeće situacije:

	R_1	R_2	R_3	R_4	R_5
K_1	3	4	4	5	●
K_2	4	4	5	5	∅
K_3	●	4	2	∅	∅
K_4	6	6	5	2	∅
K_5	2	●	∅	2	∅

Prilikom izvođenja prethodnog koraka, pretpostavljamo da smo negdje "zapisali" u kojim redovima i kolonama se nalaze nezavisne nule, tako da to ne trebamo svaki put ponovo tražiti. Prelazimo na korak 4.1. Sada je potrebno označiti neki red u kojem nema nezavisne nule. Neka je to red K_2. Nakon toga, označimo kolonu R_5 koja u redu K_1 ima zavisnu nulu, a nakon toga i red K_1 koji u koloni R_5 ima nezavisnu nulu. U ovom trenutku *ne možemo nastaviti dalje sa označavanjem*, jer u redovima K_1 i K_2 nema niti jedna nula u nekoj neoznačenoj koloni. Situacija u ovom trenutku je sljedeća:

	R_1	R_2	R_3	R_4	R_5	
K_1	3	4	4	5	●	←
K_2	4	4	5	5	∅	←
K_3	●	4	2	∅	∅	
K_4	6	6	5	2	∅	
K_5	2	●	∅	2	∅	
					↑	

Nastavljamo sa korakom 5. Potrebno je naći najmanji element Δ među vrijednostima izravnavajućih dualnih promjenljivih koje pripadaju *označenim redovima* i *neoznačenim kolonama*. To je očigledno $\Delta = 3$. Sada *uvećamo* dualne promjenljive u_1 i u_2 (koje odgovaraju *označenim redovima*) za Δ, a *umanjimo* dualnu promjenljivu v_5 (koja odgovara *označenoj koloni*) za Δ, tako da će nove vrijednosti dualnih promjenljivih biti $u_1 = 3$, $u_2 = 3$, $u_3 = 0$, $u_4 = 0$, $u_5 = 3$, $v_1 = -8$, $v_2 = -10$, $v_3 = -9$, $v_4 = -6$ i $v_5 = -3$. Nove vrijednosti izravnavajućih dualnih promjenljivih (koje naravno ne treba računati) prikazane su u sljedećoj tabeli (zajedno sa već postojećim oznakama redova i kolona):

	R_1	R_2	R_3	R_4	R_5	
K_1	∅	1	1	2	●	←
K_2	1	1	2	2	∅	←
K_3	●	4	2	∅	3	
K_4	6	6	5	2	3	
K_5	2	●	∅	2	3	
					↑	

Nova nula koja se pri tome pojavila (u presjeku reda K_1 i kolone R_1) tretira se kao *zavisna*. Primijetimo da je došlo do *manjih izmjena* u koraku 6 nego prilikom primjene "ručne verzije" algoritma, zbog činjenice da smo krenuli od *samo jednog označenog reda*. Međutim, ono što smo propustili, *biće nadoknađeno kasnije*. Stoga *nastavljamo sa označavanjem* (nastavak je *mogući*, jer se pojavila nova nula). Označićemo kolonu R_1 koja ima zavisnu nulu u redu K_1. Nakon toga se označava red K_3 koji u koloni R_1 ima nezavisnu nulu, te kolona R_4 koja ima zavisnu nulu u redu K_3 (a nije ranije bila označena). Kako smo upravo označili kolonu koja *nema nezavisnih nula*, prelazimo na korak 4.5 u kojem treba naći *povećavajući put* koji ide *naizmjenično horizontalno i vertikalno* i *naizmjenično posjećuje zavisne i nezavisne nule*, a počinje *u koloni* R_4 (*koju smo upravo označili*) i završava *u redu* K_2 (*prvom koji smo označili*). Traženi put lako nalazimo "razmotavanjem unazad", a vidljiv je u sljedećoj tablici:

	R_1	R_2	R_3	R_4	R_5	
K_1	∅	1	1	2	●	←
K_2	1	1	2	2	∅	←
K_3	●	4	2	∅	3	←
K_4	6	6	5	2	3	
K_5	2	●	∅	2	3	
	↑				↑	

Duž ovog puta *svaku zavisnu nulu proglasimo za nezavisnu a svaku nezavisnu nulu za zavisnu*, čime dobijamo novi *veći* skup nezavisnih nula. Sada *poništavamo sva označavanja* i vraćamo se na korak 4.1. Označićemo red K_4 kao jedini koji nema nezavisnih nula. Već sad nismo u stanju nastaviti dalje sa označavanjem, jer red K_4 ne sadrži niti jednu zavisnu nulu, tako da treba preći na korak 5. U ovom trenutku, situacija je kao u sljedećoj tablici:

	R_1	R_2	R_3	R_4	R_5	
K_1	●	1	1	2	∅	
K_2	1	1	2	2	●	
K_3	∅	4	2	●	3	
K_4	6	6	5	2	3	←
K_5	2	●	∅	2	3	

U koraku 5 nalazimo najmanji element Δ među vrijednostima izravnavajućih dualnih promjenljivih koje pripadaju označenim redovima i neoznačenim kolonama (tj. samo u redu K_4). To je očigledno $\Delta = 2$. Jedina dualna promjenljiva koju nakon toga treba promjeniti je u_4 koju treba *uvećati* za Δ, tako da ćemo sada imati $u_4 = 2$. Nove vrijednosti izravnavajućih dualnih promjenljivih prikazane su u sljedećoj tabeli:

	R_1	R_2	R_3	R_4	R_5	
K_1	●	1	1	2	∅	
K_2	1	1	2	2	●	
K_3	∅	4	2	●	3	
K_4	4	4	3	∅	1	←
K_5	2	●	∅	2	3	

Nastavljamo sa označavanjem. Red K_4 je sada dobio nulu u koloni R_4 koju ćemo označiti. Slijedi označavanje reda K_3 koji u koloni R_4 ima nezavisnu nulu, a zatim kolone R_1 koja u redu K_3 ima zavisnu nulu. Nakon toga, redom slijedi označavanje reda K_1, kolone R_5, te reda K_2. Dalje označavanje je nemoguće, jer nema zavisnih nula u redu K_2. U ovom trenutku, situacija je kao u sljedećoj tablici:

	R_1	R_2	R_3	R_4	R_5	
K_1	●	1	1	2	∅	←
K_2	1	1	2	2	●	←
K_3	∅	4	2	●	3	←
K_4	4	4	3	∅	1	←
K_5	2	●	∅	2	3	
	↑			↑	↑	

Ponovo smo na koraku 5. Sada je Δ najmanji element među vrijednostima izravnavajućih dualnih promjenljivih u označenim redovima K_1, K_2, K_3 i K_4 i neoznačenim kolonama R_2 i R_3. Ovaj put je $\Delta = 1$. Dualne promjenljive u_1, u_2, u_3 i u_4 se povećavaju za Δ, dok se v_1, v_4 i v_5 umanjuju za Δ. Nove vrijednosti dualnih promjenljivih iznose $u_1 = 4$, $u_2 = 4$, $u_3 = 1$, $u_4 = 3$, $u_5 = 3$, $v_1 = -9$, $v_2 = -10$, $v_3 = -9$, $v_4 = -7$ i $v_5 = -4$, dok su nove vrijednosti izravnavajućih dualnih promjenljivih prikazane u sljedećoj tabeli:

	R_1	R_2	R_3	R_4	R_5	
K_1	●	∅	∅	2	∅	←
K_2	1	∅	1	2	●	←
K_3	∅	3	1	●	3	←
K_4	4	3	2	∅	1	←
K_5	2	●	∅	3	4	
	↑			↑	↑	

Kako su se pojavile nove nule, moguće je nastaviti sa označavanjem. Možemo označiti kolonu R_2, nakon čega slijedi označavanje reda K_5, a zatim i kolone R_3, Ova kolona ne sadrži nezavisne nule, pa je vrijeme da se pronađe povećavajući put. Trenutna situacija kao i odgovarajući povećavajući put prikazani su u sljedećoj tablici:

	R_1	R_2	R_3	R_4	R_5	
K_1	●	∅	∅	2	∅	←
K_2	1	∅	1	2	●	←
K_3	∅	3	1	●	3	←
K_4	4	3	2	∅	1	←
K_5	2	●	∅	3	4	
	↑	↑		↑	↑	

Nakon razmjene statusa nezavisnih i zavisnih nula duž pronađenog povećavajućeg puta, dolazimo do situacije prikazane u sljedećoj tablici:

	R_1	R_2	R_3	R_4	R_5
K_1	∅	∅	●	2	∅
K_2	1	∅	1	2	●
K_3	●	3	1	∅	3
K_4	4	3	2	●	1
K_5	2	●	∅	3	4

S obzirom da više nemamo redova koji nemaju nezavisnih nula, optimalno rješenje je *nađeno* i može se očitati iz ove tablice. Može se uočiti da se dobija *isto optimalno rješenje* i ista završna tablica izravnavajućih dualnih promjenljivih kao što smo dobili primjenom "ručne verzije" algoritma.

Razmotrimo sada kako se može poboljšati korak 5, odnosno postići da se on izvodi u vremenu reda n a ne n^2. Za tu svrhu, treba uvesti *pomoćne veličine* s_j, $j = 1 .. n$ pri čemu svaka od veličina s_j za neko konkretno j predstavlja *najmanji od onih elemenata* j-te *kolone matrice* **D** *koji leže u označenim redovima* (tj. najmanji među vrijednostima $d_{i,j} = c_{i,j} - u_i - v_j$ za one indekse i koji odgovaraju označenim redovima). Tada je očigledno Δ jednak *najmanjoj od vrijednosti* s_j za one indekse j koji odgovaraju *neoznačenim kolonama*. Slijedi da ukoliko su veličine s_j poznate, Δ možemo izračunati u vremenu reda n. Sad samo treba obezbijediti da ove veličine *budu poznate* u trenutku kada dođemo do koraka 5. To nije teško postići. Nakon što izvršimo korak 4.1, imamo samo jedan označen red, tako da je tada prosto $s_j = d_{r,j}$ (za sve $j = 1 .. n$) gdje je r označeni red, što se izvršava u vremenu reda n. Dalje, kada god u koraku 4.4 *označimo novi red*, vrijednosti s_j se mogu *promijeniti* (jer je u igri novi red), tako da vrijednosti s_j treba *popraviti* (tj. ukoliko je u novooznačenom i-tom redu vrijednost $d_{i,j}$ manja od trenutne vrijednosti s_j, nova vrijednost s_j postaje $d_{i,j}$). Ove popravke uzimaju vrijeme reda n. Konačno, vrijednosti s_j će se promijeniti i svaki put kada se izvrši korak 6. S obzirom da ovaj korak dovodi do *smanjenja za* Δ svih elemenata $d_{i,j}$ koji leže u označenim redovima i neoznačenim kolonama, slijedi da za Δ treba *umanjiti* i sve vrijednosti s_j *koje odgovaraju neoznačenim kolonama*. I ovo se izvšava u vremenu reda n. Na ovaj način smo izbjegli kvadratično vrijeme izvršavanja koraka 5.

Ostaje još jedino da vidimo kako efektno izvršavati korak 4.2. Na prvom mjestu, jasno je da je sa ciljem da se *izbjegnu nepotrebna pretraživanja* u svim koracima od 4.1 do 4.5 neophodno koristiti i ažurirati neke strukture podataka koje će nam pomoći da do traženih informacija dođemo *odmah*. Na primjer, nedopustivo je da svaki put *tražimo* gdje se u određenom redu ili određenoj koloni nalazi nezavisna nula, nego je potrebno imati strukture podataka koje će omogućiti da se na osnovu indeksa reda (ili kolone) indeks kolone (ili reda) u kojem se nalazi odgovarajuća nula dobije odmah, bez ikakve pretrage. Trik koji omogućava ubrzanje koraka 4.2 je da za svaku kolonu koja nije označena imamo brzo dostupnu informaciju o nekom označenom redu koji u toj koloni ima zavisnu nulu, ako takvih uopće ima. Početne informacije ovog tipa se lako inicijaliziraju u koraku 4.1, a njihovo *ažuriranje* vršimo kad god se dogodi sa se kreira neka *nova nezavisna nula* (što se može dogoditi jedino u koraku 6) ili *kada u igru uđe novi red*. Slijedi da se ažuriranje ovih informacija vrši *kad god se ažuriraju i veličine* s_j. Na taj način, izvođenje koraka 4.2 možemo obavljati *kolonu po kolonu*, pri čemu pri analizi svake kolone odmah imamo dostupnu informaciju ima li u toj koloni zavisne nule u označenim redovima ili ne (ako je ima, odmah ćemo dobiti informaciju i *gdje se ona nalazi*, što će nam trebati u koracija 4.4 ili 4.5). Razumije se da traženje odgovarajuće kolone u koraku 4.2 ne treba svaki put započinjati iznova od prve kolone, nego je samo potrebno pretraživati *one kolone koje ranije nisu bile testirane* (o čemu je također potrebno voditi evidenciju u nekom spisku, pri čemu se taj spisak *proširuje* ukoliko se kreira nova zavisna nula u koraku 6).

Ukoliko se sve ovo što je gore rečeno izvede kako treba (dakle, bez nepotrebnih usporavanja), nije teško provjeriti da *ukupan broj operacija u svakoj iteraciji ne prelazi iznos proporcionalan sa n*. Kako je ukupan broj iteracija ograničen odozgo sa n^2, slijedi da se implementacija mađarskog algoritma uz sve prethodno pobrojane sugestije *može izvesti u vremenu koje ne prelazi iznos proporcionalan sa* n^3. Još jednom treba napomenuti da ovo nije i maksimalna brzina koja se može postići, s obzirom da se uz upotrebu struktura podataka zasnovanih na redovima sa prioritetom sa logaritamskim pristupom može (uz mnogo komplikovaniju implementaciju) postići vrijeme izvršavanja reda $n^2 \log_2 n$.