UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

MAGISTRSKO DELO

RAZVOJ PROGRAMSKEGA ORODJA ZA PRIMERJAVO STANDARDNE SIMPLEKS METODE IN METODE POTISNI-POVLECI

Ljubljana, september 2016 GAŠPER CIMPERMAN

IZJAVA O AVTORSTVU

Podpisani Gašper Cimperman, študent Ekonomske fakultete Univerze v Ljubljani, avtor predloženega dela z naslovom Razvoj programskega orodja za primerjavo standardne simpleks metode in metode potisni-povleci, pripravljenega v sodelovanju s svetovalko, izr. prof. dr. Liljano Ferbar Tratar,

IZJAVLJAM,

1. da sem predloženo delo pripravil/–a samostojno;

2. da je tiskana oblika predloženega dela istovetna njegovi elektronski obliki;

3. da je besedilo predloženega dela jezikovno korektno in tehnično pripravljeno v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani, kar pomeni, da sem poskrbel/–a, da so dela in mnenja drugih avtorjev oziroma avtoric, ki jih uporabljam oziroma navajam v besedilu, citirana oziroma povzeta v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani;

4. da se zavedam, da je plagiatorstvo – predstavljanje tujih del (v pisni ali grafični obliki) kot mojih lastnih – kaznivo po Kazenskem zakoniku Republike Slovenije;

5. da se zavedam posledic, ki bi jih na osnovi predloženega dela dokazano plagiatorstvo lahko predstavljalo za moj status na Ekonomski fakulteti Univerze v Ljubljani v skladu z relevantnim pravilnikom;

6. da sem pridobil/–a vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v predloženem delu in jih v njem jasno označil/–a;

7. da sem pri pripravi predloženega dela ravnal/–a v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobil/–a soglasje etične komisije;

8. da soglašam, da se elektronska oblika predloženega dela uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;

9. da na Univerzo v Ljubljani neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve predloženega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja predloženega dela na voljo javnosti na svetovnem spletu preko Repozitorija Univerze v Ljubljani;

10. da hkrati z objavo predloženega dela dovoljujem objavo svojih osebnih podatkov, ki so navedeni v njem in v tej izjavi.

V Ljubljani, dne _____ Podpis študenta: _____

KAZALO

KAZALO TABEL

KAZALO SLIK

UVOD

Padberg je v svojem delu zapisal, da so končni sistemi linearnih enačb in neenačb s končnim številom spremenljivk v svoji matematični osnovi bistvo linearnega programiranja. Zgodnja prototipa linearnega programa sta bila skoraj simultano objavljena v Združenih državah Amerike (1941) in Sovjetski zvezi (1939) in sta bila navidez konstruirana neodvisno en od drugega, vendar sta se nevede ukvarjala s problemom, ki se ga je lotil že francoski matematik Gaspard Monge (1746–1818) v svojem delu *Memoire sur la theorie des deblais et des remblais* leta 1781. Drugi splošno znani prototipi vsebujejo model linearnega programiranja za npr. iskanje minimuma stroškov pri oblikovanju diete ali problema skladišča, pri katerem se ob sezonskih nihanjih stroškov in prihodkov išče optimalna politika nakupovanja, skladiščenja in prodaje nekega blaga, na primer žita. Kar se tiče zgodovinskih začetkov, se je morda že upravnik Asirske žitne kašče v davnem desetem stoletju pred našim štetjem igral s podobno idejo (Padberg, 1999).

Geary in Spencer (1973) ugotavljata, da nam ob trditvi, da je tehnike linearnega programiranja mogoče uporabiti v večini primerov, pri katerih se nekdo sooči z množico postopkov in tehnik, izmed katerih želi izbrati najboljšo glede na standard (npr. maksimiranje dobička ali minimiziranje stroškov) – pri tem pa mora upoštevati določene omejitve (npr. da določeno kmetijsko zemljišče ne sme presegati določene površine) – postane približno jasno, kako široko področje obravnave je linearno programiranje.

Najbolj znan in najširše uporabljan postopek za reševanje problemov linearnega programiranja se imenuje simpleks metoda, ki jo je razvil George Dantzig leta 1947. Izraz "simpleks" ni v nikakršni povezavi z metodo, kot je v uporabi danes, temveč s posebnim problemom, ki so ga preučevali v zgodnjih fazah razvoja metode (Hadley, 1973).

Simpleks metode so osredotočene na iskanje dopustnih rešitev, ki jim ustreza enolično določena ekstremna točka konveksnega poliedra možnih rešitev in še določena baza vektorskega prostora, ki sestoji iz *m* linearno neodvisnih vektorjev. V bazi nato zamenjamo kak njen vektor s takšnim, ki ga v bazi še ni, dobimo novo bazo in njej ustrezno ekstremno točko, ki ji ustreza nova vrednost kriterijske funkcije. Pri tem se premikamo po mejah področja dopustnih rešitev (Vadnal, 1977).

Čeprav je splošna simpleks metoda z različnimi variantami izredno popularna in tudi učinkovita, pa obstajajo problemi linearnega programiranja, ki jih z eno od teh metod v sprejemljivem času ni mogoče rešiti. Poleg problema zasedenosti kapacitet računalnika (predvsem spomina) pri reševanju problemov linearnega programiranja, je eden od glavnih kriterijev ustreznosti oz. učinkovitosti metode tudi kriterij povprečnega časa izvajanja algoritma, ki se v praksi prevede na analizo povprečnega števila elementarnih aritmetičnih operacij in števila korakov pivotiranja tabele. Zato je pri praktični uporabi s pomočjo

natančne analize povprečnega obnašanja algoritma dobro ugotoviti, katera varianta najbolj učinkovito rešuje določeno vrsto problema (Borgwardt, 1987).

Simpleks algoritem pri reševanju problemov linearnega programiranja (v nadaljevanju problemi LP) zahteva uvedbo umetnih spremenljivk, katerim primanjkuje osnovna rešljivost v začetni točki. Zato je bil predstavljen nov algoritem splošne uporabe, imenovan potisni-povleci (angl. *Push–Pull*), ki se izogiba rabi umetnih spremenljivk (Arsham, Damij, & Grad, 2003).

Algoritem poleg pripravljalne vsebuje dve glavni fazi. Prva faza, "Potisni" (angl. *Push*), zgradi bazo, ki je lahko rešljiva ali pa ne. Za razliko od splošne simpleks in dualne simpleks metode se lahko začne izvajanje algoritma z nepopolno bazo. Ko je baza zgrajena, potiska faza "Potisni" rešitev proti optimumu, pri čemer je kriterij za ugotavljanje optimuma podoben tistemu iz splošne simpleks metode. Pri tem se lahko zgodi, da iterativni proces reševanja uide iz območja rešljivosti. V tem primeru nastopi faza "Povleci" (angl. *Pull*), ki z uporabo pravil, podobnih pravilom dualne simpleks metode, vrne izvajanje algoritma nazaj v območje rešljivosti (Arsham, Damij, & Grad, 2003).

Namen magistrskega dela je na kratko predstaviti problematiko linearnega programiranja in v okviru tega standardno simpleks ter novejšo metodo potisni-povleci, nato pa razviti programsko orodje za reševanje problemov LP po obeh metodah in izpeljati primerjavo njune učinkovitosti s ciljem preverjanja Arshamove teze o boljši učinkovitosti novejše metode (Arsham, 1997).

Uporabljena metodologija obsega študijo obstoječih virov in literature, na podlagi katere sem razvil programsko orodje SixPap. Sledi analiza s tem orodjem pridobljenih parametrov izvajanja obeh algoritmov na 15 primerih problemov LP z interpretacijo dobljenih rezultatov.

Delo je razdeljeno na štiri poglavja, pri čemer prvi dve predstavljata teoretični, drugi dve pa praktični del. V prvih dveh poglavjih sta tako na podlagi študije virov in literature podrobno predstavljeni obe metodi.

V nadaljevanju je prikazana računalniška implementacija obeh metod in razvoj programskega orodja SixPap, ki služi za reševanje problemov LP. Pri reševanju je možno uporabiti eno, drugo ali obe metodi istočasno, vgrajen ima tudi mehanizem za merjenje in primerjavo učinkovitosti obeh algoritmov (Arsham, Cimperman, Damij N., Damij T., & Grad, 2005).

V zadnjem delu je s tem orodjem izvedena primerjalna analiza obeh metod na čim večjem številu numeričnih primerov, z namenom ugotoviti dobre in slabe lastnosti metod ter prednosti in slabosti ene metode v primerjavi z drugo.

Merjenje učinkovitosti izvajanja deloma temelji na parametrih, ki so jih za primerjavo uporabili Arsham, Baloh, Damij in Grad (2003), dopolnjenimi z merjenjem števila prehodov zank, števila odločitvenih stavkov ter z ugotavljanjem pojava degeneracije. Merjenje je implementirano v predstavljenem programskem orodju SixPap in se izvaja avtomatično.

Orodje poleg samega merjenja omogoča tabelarično primerjavo rezultatov izvajanja obeh metod na večjem številu problemov naenkrat. Primerjava zajema prikaz vrednosti opazovanih parametrov in prikaz izračuna razlik med njimi (Arsham, Cimperman, Damij N., Damij T., & Grad, 2005).

Sicer je na trgu kar nekaj orodij za reševanje problemov LP, kot so npr.:

1. *Linear Program Solver*, ki ga najdemo na spletnih straneh odprtokodnih projektov *SourceForge* (*Linear Program Solver*, 2016),
2. *LINDO™*, ki ga najdemo na spletnih straneh podjetja Lindo Systems Inc. (*LINDO™ software products*, 2016),
3. *GLPK*, ki ga najdemo na spletnih straneh projekta *GNU* (*GLPK - GNU Linear Programming Kit)*, 2016) ali
4. *AIMMS*, ki ga najdemo na spletnih straneh podjetja AIMMS B.V. (*AIMMS Linnear Programming*, 2016),

vendar nobeno ne izvaja direktne primerjave standardne simpleks in potisni-povleci metode.

Razvito orodje je na splošno možno uporabiti za reševanje problemov LP in nadaljnjo analizo poteka izvajanja obeh algoritmov. Glede na podroben izpis poteka izvajanja se lahko orodje uporabi tudi za študijo delovanja ene ali druge metode. Glede na modularno zasnovo se lahko orodju dodajo tudi druge metode, s čimer se razširi analizo učinkovitosti in delovanja.

1 STANDARDNA SIMPLEKS METODA

Leta 1947 je George Dantzig predstavil simpleks metodo (angl. *Simplex Method*) za reševanje širokega spektra optimizacijskih problemov, znanih kot problemov LP. Zavedati se moramo, da je izraz "programiranje" v 40. letih 20. stoletja pomenil načrtovanje in ne pisanje računalniške programske kode. Pri tem se je Dantzig opiral na dognanja in prispevke Koopmansa, Nobelovega nagrajenca na področju ekonomike, ter velikega matematika Johna von Neumanna, ki je linearno programiranje povezal s teorijo iger (Feiring, 1986).

Feiring (1986) v svojem uvodu zapiše še, da področje linearnega programiranja spada v sklop matematičnega programiranja, to pa spada v širše področje operacijskih raziskav. Operacijske raziskave imajo sledeče značilnosti:

1. Obstoj alternativ ali opcij.
2. Rešitve se merijo glede na doseganje specifičnih ciljev ali kriterijev.
3. Vključujejo optimizacijo – izbor najboljše alternative glede na dane pogoje.
4. Vključujejo sistemski pogled, ki obravnava medsebojne relacije komponent in ne posamezne komponente.

V številnih študijah uporabe metod operacijskih raziskav je linearno programiranje na prvem ali drugem mestu, ugotavlja Feiring (1986). Druge tehnike operacijskih raziskav so še: simulacije, mrežne analize, teorija vrst, stohastični procesi, dinamično programiranje, regresijska analiza, nelinearno programiranje in teorija iger. Linearno programiranje se uporablja za reševanje gospodarskih, socialnih, vojaških, proizvodnih problemov in problemov z drugih področij.

Feiring (1986) opredeli linearno programiranje kot posebno področje matematičnega programiranja, ki se ukvarja z učinkovito alokacijo omejenih virov znanim aktivnostim, z namenom doseganja želenih ciljev, kot sta maksimiziranje dobička ali minimiziranje stroškov. Linearnost nekaterih modelov je zagotovljena že na osnovi fizičnih lastnosti problema, nekatere nelinearne modele pa je mogoče pretvoriti v linearne z uporabo ustreznih matematičnih transformacij. Da problem lahko obravnavamo z linearnim programiranjem, mora zadoščati naslednjim trem pogojem:

1. Spremenljivke, ki nastopajo pri odločitvi, morajo biti nenegativne (večje ali enake nič).
2. Namenska funkcija za iskanje optimalnih vrednosti je linearna funkcija odločitvenih spremenljivk.
3. Pogoje oz. omejitve, ki vladajo procesu, je možno izraziti z linearnimi enačbami ali neenačbami odločitvenih spremenljivk.

Za široko uporabo linearnega programiranja navaja Fiering (1986) naslednje tri glavne razloge:

1. Veliko število problemov z različnih področij je možno zapisati ali aproksimirati kot modele linearnega programiranja.
2. Na razpolago je kar nekaj učinkovitih metod za reševanje problemov linearnega programiranja.
3. Analiza občutljivosti omogoča variiranje podatkov problema, kot nadgradnja metod iskanja rešitve problema LP.

Kot zaključno ugotovitev v uvodu Fiering navede dejstvo, da so metode reševanja problemov LP iterativne, zato za reševanje večine resnejših problemov potrebujejo računalnik. Z razvojem računalniške strojne opreme in ustreznih programskih orodij postaja reševanje obsežnih problemov LP učunkovito, hitro in poceni (Feiring, 1986).

Večina odločitev s področja operacij stremi k čim boljšemu izkoristku virov organizacije. Ti viri so navadno: stroji, delovna sila, finance, čas, skladiščni prostor ali surovine, uporabijo se lahko za proizvodnjo izdelkov, kot so stroji, pohištvo, hrana ali oblačila, ali za opravljanje storitev, kot so urniki pošiljanj in prizvodnje, oglaševalske politike ali investicijske odločitve. Pri alokaciji virov si odločevalci s področja proizvodnje in operacij v veliki meri pomagajo z linearnim programiranjem – matematično tehniko, razvito posebej za ta namen (Heizer & Render, 1993).

Za boljšo ilustracijo uporabnosti je v nadaljevanju navedenih nekaj primerov s področja upravljanja operacij, pri katerih se uspešno uporabljajo metode linearnega programiranja. To so (Heizer & Render, 1993):

1. priprava proizvodnega načrta, ki bo zadovoljil bodoče povpraševanje po proizvodih in hkrati minimiziral skupne proizvodne in skladiščne stroške,
2. izbor proizvodnega asortimana z najboljšim izkoristkom strojev in delovne sile ob maksimizaciji dobička podjetja,
3. določitev kakovosti surove nafte za maksimiziranje dobička,
4. izbor različnih mešanic surovin v proizvodnji krmil za pripravo končnih krmil ob minimizaciji stroškov,
5. priprava distribucijskega sistema, ki ima najnižje skupne stroške transporta iz različnih skladišč na različne lokacije,
6. priprava načrta vožnje šolskih avtobusov z minimalno dnevno prevoženimi kilometri pri pobiranju in odlaganju učencev,
7. razporeditev policijskih patrulj na področja povečane kriminalne aktivnosti z namenom minimizacije odzivnih časov na urgentne klice.

Veliko zanimivih praktičnih primerov lahko najdemo tudi v delih Bradleyja, Haxa in Magnantija (1977), Bazarae, Jarvisa in Sheralija (1977), Hillierja in Liebermana (1977), Winstona (2004) ter Curwina, Slaterja in Eadsona (2013).

V nadaljevnaju je razlaga algoritma povzeta po dispoziciji algoritma za reševanje problema po simpleks metodi (Vadnal, 1986).

1.1 Algoritem

Za izvajanje algoritma po standardni simpleks metodi vzamemo problem linearnega programiranja v njegovi najbolj splošni matematični formulaciji, ki se glasi:

Določiti je treba vrednosti spremenljivk

$$X_1,...,X_s,$$

ki zadoščajo pogojem nenegativnosti:

$$X_1,...,X_s \geq 0 \tag{1}$$

in linearnim enačbam ali neenačbam:

$$\begin{aligned} &a_{11}X_1 + ... + a_{1s}X_s \ \begin{matrix}\geq \\ < \end{matrix}\ b_1 \\ &... \\ &a_{m1}X_1 + ... + a_{ms}X_s \ \begin{matrix}\geq \\ < \end{matrix}\ b_m, \end{aligned} \tag{2}$$

tako, da ima namenska funkcija:

$$f(X_1,...,X_s) = c_1X_1 + ... + c_sX_s \tag{3}$$

pri danih vrednostih spremenljivk minimum oziroma maksimum.

V nadaljevanju sledi opis algoritma po korakih.

1.1.1 Predpriprava

Zaradi poenostavitve algoritma pretvorimo problem iskanja maksimuma v problem iskanja minimuma tako, da namensko funkcijo pomnožimo s številom –1.

Poiščemo ustrezno vrednost za veliki M. Naj bo na primer s faktorjem 10 pomnožena absolutna vrednost največjega koeficienta namenske funkcije.

1.1.2 Korak 1: razširi sistem neenačb v sistem enačb

Z uvedbo dopolnilnih, dodatnih in umetnih spremenljivk pretvorimo vse neenačbe (razen pogoja nenegativnosti) v enačbe po naslednjih pravilih:

1. V primeru neenačbe tipa manjše/enako (≤; t. i. omejitev virov) vpeljemo **dopolnilno** spremenljivko z vrednostjo koeficienta +1, vrednost koeficienta namenske funkcije pri tej spremenljivki je enaka nič.
2. V primeru neenačbe tipa večje/enako (≥; t. i. produkcijska omejitev) vpeljemo **dodatno** spremenljivko z vrednostjo koeficienta –1, vrednost koeficienta namenske funkcije pri tej spremenljivki je enaka nič. Uvedemo tudi **umetno** spremenljivko s koeficientom +1 in vrednostjo koeficienta v namenski funkciji veliki M. Veliki M naj predstavlja dovolj veliko število.
3. V primeru enačbe uvedemo **umetno** spremenljivko s koeficientom +1 in vrednostjo koeficienta v namenski funkciji veliki M.

1.1.3 Korak 2: konstruiraj začetno tabelo

Začetna tabela zajema matriko koeficientov pogojnih enačb, ki ji je (na konec) dodana vrstica z diferencami $Z_j - C_j$, ki jih izračunamo po naslednjem obrazcu:

$$Z_j - C_j = \left(\sum_{i=1}^{m} a_{ij} \cdot c_{BVS(i)} \right) - c_j , \tag{4}$$

pri čemer pomeni:

1. m – število pogojnih neenačb,
2. a_{ij} – koeficient *i*-te pogojne neenačbe pri *j*-ti spremenljivki,
3. $c_{BVS(i)}$ – koeficient namenske funkcije pri spremenljivki, ki je v množici baznih vektorjev (angl. *Basic Variable Set*, v nadaljevanju *BVS*) na *i*-tem mestu,
4. c_j – koeficient namenske funkcije pri *j*-ti spremenljivki.

Na konec je dodan še stolpec, ki vsebuje vrednosti desnih strani neenačb. V začetni *BVS* dodamo vse dopolnilne in umetne spremenljivke. Dimenzija *BVS* bo tako enaka *m* – številu pogojnih neenačb. Dimenzija tabele pa bo enaka *m + 1* vrstic in *s + d + 1* stolpcev, pri čemer *s* pomeni število originalnih, *d* število dodatnih, dopolnilnih in umetnih spremenljivk, njihovo vsoto pa označimo z *n*.

1.1.4 Korak 3: poišči nov bazni vektor

Med diferencami $Z_j - C_j$ poiščemo največjo pozitivno vrednost. Pripadajoči bazni vektor bo nov bazni vektor. Stolpec, v katerem se nahaja, postane pivotni stolpec in ga označimo s *k*.

1.1.5 Korak 4: poišči bazni vektor, ki bo izstopil

Poišči tisti vektor v *BVS*, kateremu pripada najmanjša vrednost količnika vrednosti desnih strani neenačb (angl. *right hand side*, v nadaljevanju *RHS*) in vrednosti koeficienta pri vstopnem vektorju (angl. *Column Ratio*, v nadaljevanju *CR*). Pri izračunu upoštevamo samo pozitivne komponente novega vektorja (stolpec *k*). Izbrano vrstico označimo z *r*. Količnik *CR* izračunamo s pomočjo izraza:

$$CR(i,k) = \frac{RHS(i)}{A(i,k)}; A(i,k) > 0 , \tag{5}$$

pri čemer so:

1. $CR(i,k)$ – *i*-ti količnik *k*-tega stolpca, iščemo najmanjšega pozitivnega,
2. i – obravnavana vrstica,

3. k – v prejšnjem koraku izbrani stolpec,
4. $RHS(i)$ – vrednost desne strani *i*-te neenačbe,
5. $A(i,k)$ – element matrike koeficientov pogojnih neenačb z indeksoma *i* in *k*.

1.1.6 Korak 5: generiraj novo tabelo s parametroma *k* in *r*

Novo tabelo generiramo s pomočjo transformacijskih pravil, in sicer:

1. **Koeficiente v izbrani vrstici** *r* delimo s ključnim koeficientom *A(r,k)*:

$$A'(r,\, j) = \frac{A(r,\, j)}{A(r,k)};\; j : 1..n+1, \tag{6}$$

pri čemer so:

a) $A'(r, j)$ – novi element tabele z indeksoma *r* in *j*,
b) $A(r, j)$ – stari element tabele z indeksoma *r* in *j*,
c) $A(r, k)$ – ključni element,
d) *r* – v koraku 3 določena vrstica,
e) *k* – v koraku 4 določeni stolpec,
f) *j* – indeks stolpcev, ki teče od 1 do *n* + 1,
g) *n* – število vseh spremenljivk.

Pri tem predstavlja stolpec *n* + 1 vrednost *RHS* izbrane vrstice.

2. **Koeficiente v ostalih vrsticah** pa izračunamo po naslednjem obrazcu:

$$\begin{aligned} &A'(i,\, j) = A(i,\, j) - A'(r,\, j) \cdot A(i,k) \\ &i = 1,..., m+1;\; j = 1,..., n+1, \end{aligned} \tag{7}$$

pri čemer so:

a) $A'(i, j)$ – novi element tabele z indeksoma *i* in *j*,
b) $A(i, j)$ – stari element tabele z indeksoma *i* in *j*,
c) $A'(r, j)$ – novi element tabele z indeksoma *r* in *j*,
d) $A(i, k)$ – stari element tabele z indeksoma *i* in *k*,
e) *r* – v koraku 3 določena vrstica,
f) *k* – v koraku 4 določeni stolpec,
g) *j* – indeks stolpcev, ki teče od 1 do *n* + 1,
h) *n* – število vseh spremenljivk,
i) *i* – indeks vrstic, ki teče od 1 do *m* + 1, pri čemer preskoči *r*-to vrstico,
j) *m* – število pogojnih neenačb.

Ker v obrazcu (7) uporabljamo novo vrednost elementa *A'(r,k)*, moramo biti pozorni na to, da točko 1 izvedemo pred točko 2. Zadnja vrstica (m + 1) predstavlja diference $Z_j - C_j$.

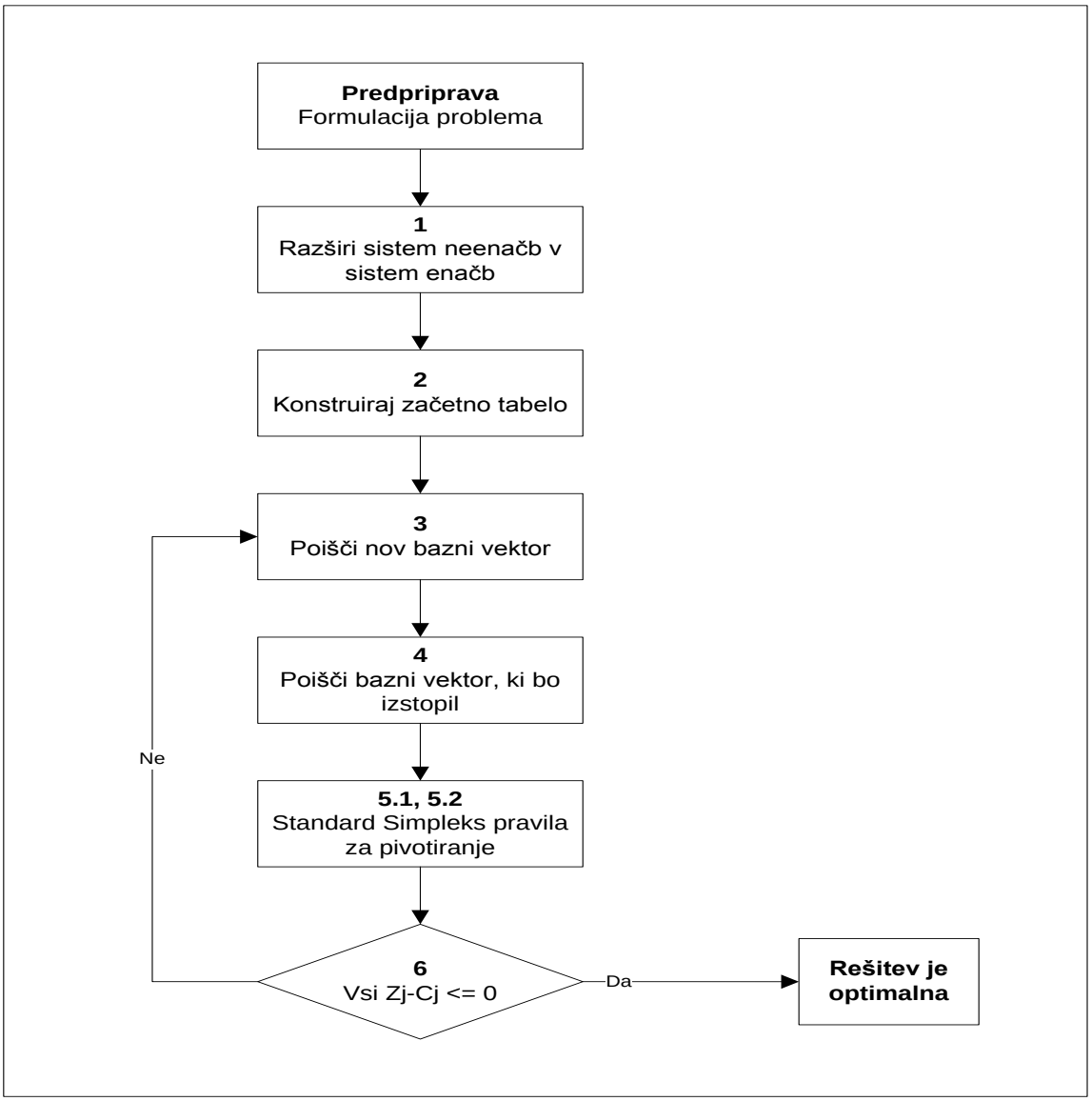
1.1.7 Korak 6: preveri trenutno rešitev

Če za vse $\boldsymbol{Z_j - C_j}$ velja, da so **manjši ali enaki nič** $\forall(Z_j - C_j) \leq 0$, pomeni, da je trenutna rešitev **optimalna** in izboljšava ni več mogoča, sicer se vrnemo na **korak 3**.

1.2 Diagram poteka algoritma

Potek izvajanja algoritma simpleks je prikazan na diagramu, ki ga prikazuje Slika 1.

Slika 1: Diagram poteka izvajanja algoritma po simpleks metodi



Vir: Povzeto po A. Vadnal, Linearno programiranje, 1986, str. 170-172.

1.3 Primeri reševanja problemov

V nadaljevanju je prikazana uporaba algoritma standardne simpleks metode pri iskanju optimalnih rešitev na primeru dveh zgledov. Oba zgleda sta povzeta po članku *"A computer implementation of the Push-and-Pull algorithm and its computational comparison with LP simplex method"* (Arsham, Cimperman, Damij N., Damij T., & Grad, 2005).

1.3.1 Zgled 1

Iščemo maksimum namenske funkcije:

$$2X_1 + 6X_2 + 8X_3 + 5X_4, \tag{8}$$

ki zadošča naslednjim pogojem:

$$\begin{aligned} &4X_1 + X_2 + 2X_3 + 2X_4 \geq 80 \\ &2X_1 + 5X_2 + 4X_4 \leq 40 \\ &2X_2 + 4X_3 + X_4 = 120 \\ &X_1, X_2, X_3, X_4 \geq 0. \end{aligned} \tag{9}$$

Predpriprava. Ker gre za maksimizacijski problem, namensko funkcijo pomnožimo z minus 1 in s tem problem prevedemo na iskanje minimuma. Nova namenska funkcija se tako glasi:

$$-2X_1 - 6X_2 - 8X_3 - 5X_4. \tag{10}$$

Poiščimo minimum te funkcije.

Določimo vrednost za veliki *M*. Naj bo z 10 pomnožena absolutna vrednost največjega koeficienta, v našem primeru torej ***M* = 80**.

Ostali pogoji zadoščajo začetnim pogojem za izvajanje algoritma, saj so vse desne strani pogojnih neenačb nenegativne, isto velja za vse spremenljivke. Zato lahko nadaljujemo s prvim korakom algoritma.

Korak 1 – razširi sistem neenačb v sistem enačb. Prva neenačba predstavlja t. i. produkcijsko omejitev (tipa večje/enako), zato jo dopolnimo z dodatno spremenljivko S_5 s koeficientom –1. Pripadajoča vrednost koeficienta v namenski funkciji bo enaka nič. Uvedemo tudi umetno spremenljivko A_7 s koeficientom +1 in vrednostjo koeficienta v namenski funkciji 80.

Druga je tipa manjše/enako (predstavlja omejitev virov), zato jo dopolnimo z dopolnilno spremenljivko S_6 s koeficientom +1 in vrednostjo koeficienta v namenski funkciji nič.

Tretja je enačba, zato uvedemo umetno spremenljivko A_8 s koeficientom +1 in vrednostjo koeficienta v namenski funkciji 80.

Razširjeni sistem se tako glasi:

$$\begin{aligned}
&\min: -2X_1 - 6X_2 - 8X_3 - 5X_4 + 0S_5 + 0S_6 + 80A_7 + 80A_8 \\
&4X_1 + X_2 + 2X_3 + 2X_4 - S_5 \quad + A_7 \quad = 80 \\
&2X_1 + 5X_2 \quad + 4X_4 \quad + S_6 \quad = 40 \\
&\quad 2X_2 + 4X_3 + X_4 \quad + A_8 = 120 \\
&X_1, X_2, X_3, X_4, S_5, S_6, A_7, A_8 \geq 0.
\end{aligned} \tag{11}$$

Korak 2 – konstruiraj začetno tabelo. Vidimo, da je *m* (število pogojnih enačb) enak 3, *s* (število originalnih spremenljivk) je enak 4, *d* (število dopolnilnih, dodatnih in umetnih spremenljivk) pa je enak 4.

V razširjenem sistemu enačb nastopajo ena dopolnilna in dve umetni spremenljivki. To so:

1. A_7, ki nastopa v prvi pogojni enačbi,
2. S_6 v drugi in
3. A_8 v tretji.

V tem zaporedju zasedejo mesta v začetnem *BVS*. Zapišemo po vrsti še koeficiente pogojnih enačb, v zadnji stolpec dodamo njihove vrednosti *RHS* in na konec dodamo vrstico z diferencami $Z_j - C_j$, ki jih izračunamo po obrazcu (4) in dobimo:

Tabela 1: Metoda simpleks, zgled 1, iteracija 1, korak 2

BVS	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{X_4}$	$\mathbf{S_5}$	$\mathbf{S_6}$	$\mathbf{A_7}$	$\mathbf{A_8}$	***RHS***
A_7	4	1	2	2	–1	0	1	0	80
S_6	2	5	0	4	0	1	0	0	40
A_8	0	2	4	1	0	0	0	1	120
$Z_j - C_j$	322	246	488	245	–80	0	0	0	

Korak 3 – poišči nov bazni vektor. Iz Tabele 1 razberemo, da največja diferenca $Z_j - C_j$ pripada spremenljivki X_3, zato je ta prvi kandidat za vstop v *BVS*.

Označimo $k = 3$.

Korak 4 – poišči vektor, ki bo izstopil. S pomočjo izraza (5) izračunamo koeficiente *CR*, pri čemer je $k = 3$:

Tabela 2: Metoda simpleks, zgled 1, iteracija 1, korak 4

BVS	**...**	$\mathbf{X_3}$	**...**	***RHS***	$\mathbf{CR_{i,3}}$
A_7		2		80	40,00
S_6		0		40	–
A_8		4		120	30,00

Najmanjši *CR* pripada tretji vrstici, zato bo izstopil vektor, ki je v *BVS* na tretjem mestu. To je A_8, indeks pivotne vrstice $r = 3$.

Korak 5 – generiraj novo tabelo s parametroma *k* in *r*. S pomočjo izrazov (6) in (7) izračunamo novo tabelo, ki se glasi:

Tabela 3: Metoda simpleks, zgled 1, iteracija 1, korak 5

BVS	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{X_4}$	$\mathbf{S_5}$	$\mathbf{S_6}$	$\mathbf{A_7}$	$\mathbf{A_8}$	***RHS***
A_7	4	0	0	1,5	–1	0	1	–0,5	20
S_6	2	5	0	4	0	1	0	0	40
X_3	0	0,5	1	0,25	0	0	0	0,25	30
$Z_j - C_j$	322	2	0	123	–80	0	0	–122	

Korak 6 – preveri trenutno rešitev. Ker iz Tabele 3 razberemo, da obstajajo diference $Z_j - C_j$, ki so večje od nič, je nadaljnja izboljšava še možna. Zato ponovimo korak 3.

Korak 3 – poišči nov bazni vektor. Iz Tabele 3 razberemo, da največja diferenca $Z_j - C_j$ pripada spremenljivki X_1, zato je ta naslednji kandidat za vstop v *BVS*. Označimo $k = 1$.

Korak 4 – poišči vektor, ki bo izstopil. S pomočjo izraza (5) izračunamo koeficiente *CR*, pri čemer je $k = 1$:

Tabela 4: Metoda simpleks, zgled 1, iteracija 2, korak 4

BVS	$\mathbf{X_1}$	**...**	***RHS***	$\mathbf{CR_{i,1}}$
A_7	4		20	5
S_6	2		40	20
X_3	0		30	–

Najmanjši *CR* pripada prvi vrstici, zato bo izstopil A_7. Indeks pivotne vrstice $r = 1$.

Korak 5 – generiraj novo tabelo s parametroma *k* in *r*. S pomočjo izrazov (6) in (7) izračunamo novo tabelo, ki se glasi:

Tabela 5: Metoda simpleks, zgled 1, iteracija 2, korak 5

BVS	X_1	X_2	X_3	X_4	S_5	S_6	A_7	A_8	***RHS***
X_1	1	0	0	0,375	–0,25	0	0,25	–0,125	5
S_6	0	5,0	0	3,250	0,50	1	–0,50	0,250	30
X_3	0	0,5	1	0,250	0	0	0	0,250	30
$Z_j - C_j$	0	2,0	0	2,250	0,50	0	–80,50	–81,750	

Korak 6 – preveri trenutno rešitev. Ker iz Tabele 5*Tabela 5* razberemo, da obstajajo diference $Z_j - C_j$, ki so večje od nič, je nadaljnja izboljšava še možna. Zato ponovimo korak 3.

Korak 3 – poišči nov bazni vektor. Iz Tabele 5 razberemo, da največja diferenca $Z_j - C_j$ pripada spremenljivki X_4, zato je ta naslednji kandidat za vstop v *BVS*.

Označimo $k = 4$.

Korak 4 – poišči vektor, ki bo izstopil. S pomočjo izraza (5) izračunamo koeficiente *CR*, pri čemer je $k = 4$:

Tabela 6: Metoda simpleks, zgled 1, iteracija 3, korak 4

BVS	...	X_4	...	***RHS***	$CR_{i,4}$
X_1		0,375		5	13,333
S_6		3,250		30	9,231
X_3		0,250		30	120,000

Najmanjši *CR* pripada drugi vrstici, zato bo izstopil S_6, indeks pivotne vrstice $r = 2$.

Korak 5 – generiraj novo tabelo s parametroma *k* in *r*. S pomočjo izrazov (6) in (7) izračunamo novo tabelo, ki se glasi:

Tabela 7: Metoda simpleks, zgled 1, iteracija 3, korak 5

BVS	X_1	X_2	X_3	X_4	S_5	S_6	A_7	A_8	***RHS***
X_1	1	–0,577	0	0	–0,308	–0,115	0,308	–0,154	1,538
X_4	0	1,538	0	1	0,154	0,308	–0,154	0,077	9,231
X_3	0	0,115	1	0	–0,038	–0,077	0,038	0,231	27,692
$Z_j - C_j$	0	–1,461	0	0	0,154	–0,692	–80,154	–81,923	

Korak 6 – preveri trenutno rešitev. Ker iz Tabele 7 razberemo, da obstajajo diference $Z_j - C_j$, ki so večje od nič, je nadaljnja izboljšava še možna.

Ponovimo korak 3.

Korak 3 – poišči nov bazni vektor. Iz Tabele 7 razberemo, da največja diferenca $Z_j - C_j$ pripada spremenljivki S_5, zato je ta naslednji kandidat za vstop v *BVS*.

Označimo $k = 5$.

Korak 4 – poišči vektor, ki bo izstopil. S pomočjo izraza (5) izračunamo koeficiente *CR*, pri čemer je $k = 5$:

Tabela 8: Metoda simpleks, zgled 1, iteracija 4, korak 4

BVS	**...**	$\mathbf{S_5}$	**...**	***RHS***	$\mathbf{CR_{i,5}}$
X_1		–0,308		1,538	–
X_4		0,154		9,231	60
X_3		–0,038		27,692	–

Najmanjši *CR* pripada drugi vrstici, zato bo izstopil X_4. Indeks pivotne vrstice $r = 2$.

Korak 5 – generiraj novo tabelo s parametroma *k* in *r*. S pomočjo izrazov (6) in (7) izračunamo novo tabelo, ki se glasi:

Tabela 9: Metoda simpleks, zgled 1, iteracija 4, korak 5

BVS	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{X_4}$	$\mathbf{S_5}$	$\mathbf{S_6}$	$\mathbf{A_7}$	$\mathbf{A_8}$	***RHS***
X_1	1	2,5	0	2,0	0	0,5	0	0	20
S_5	0	10,0	0	6,5	1	2,0	–1	0,50	60
X_3	0	0,5	1	0,25	0	0	0	0,25	30
$Z_j - C_j$	0	–3,0	0	–1,0	0	–1,0	–80	–82,00	

Korak 6 – preveri trenutno rešitev. Ker iz Tabele 9 razberemo, da so vse diference $Z_j - C_j$ manjše ali enake nič, to pomeni, da nadaljnja izboljšava ni možna.

Rešitev je ***optimalna***, vrednosti spremenljivk so:

$$\boldsymbol{X_1 = 20}, \boldsymbol{X_2 = 0}, \boldsymbol{X_3 = 30} \text{ in } \boldsymbol{X_4 = 0}.$$

Ko to vstavimo v originalno namensko funkcijo, dobimo za vrednost ***280***.

1.3.2 Zgled 2

Iščemo maksimum namenske funkcije:

$$2X_1 + 3X_2 - 6X_3, \tag{12}$$

ki zadošča naslednjim pogojem:

$$\begin{aligned} &X_1 + 3X_2 + 2X_3 \leq 3 \\ &-2X_2 + X_3 \leq 1 \\ &X_1 - X_2 + X_3 = 2 \\ &2X_1 + 2X_2 - 3X_3 = 0 \\ &X_1, X_2, X_3 \geq 0. \end{aligned} \tag{13}$$

Predpriprava. Ker gre za maksimizacijski problem, namensko funkcijo pomnožimo s faktorjem –1 in s tem problem prevedemo na iskanje minimuma. Nova namenska funkcija se tako glasi:

$$-2X_1 - 3X_2 + 6X_3. \tag{14}$$

Iščemo minimum te funkcije.

Določimo vrednost za veliki *M*. Naj bo z 10 pomnožena absolutna vrednost največjega koeficienta, v našem primeru torej ***M* = 60**.

Ostali pogoji zadoščajo začetnim pogojem za izvajanje algoritma, saj so vse desne strani pogojnih neenačb nenegativne, isto velja za vse spremenljivke. Zato lahko nadaljujemo s prvim korakom algoritma.

Korak 1 – razširi sistem neenačb v sistem enačb. Prva neenačba je tipa manjše/enako (predstavlja omejitev virov), zato jo dopolnimo z dopolnilno spremenljivko S_4 s koeficientom +1 in vrednostjo koeficienta v namenski funkciji nič.

Tudi druga neenačba je tipa manjše/enako (predstavlja omejitev virov), zato jo dopolnimo z dopolnilno spremenljivko S_5 s koeficientom +1 in vrednostjo koeficienta v namenski funkciji nič.

Tretja je enačba, zato uvedemo umetno spremenljivko A_6 s koeficientom +1 in vrednostjo koeficienta v namenski funkciji 60.

Isto velja za četrto, zato uvedemo umetno spremenljivko A_7 s koeficientom +1 in vrednostjo koeficienta v namenski funkciji 60.

Razširjeni sistem se tako glasi:

$$\begin{aligned}
&\min: -2X_1 - 3X_2 + 6X_3 + 0S_4 + 0S_5 + 60A_6 + 60A_7 \\
&X_1 + 3X_2 + 2X_3 + S_4 = 3 \\
&-2X_2 + X_3 + S_5 = 1 \\
&X_1 - X_2 + X_3 + A_6 = 2 \\
&2X_1 + 2X_2 - 3X_3 + A_7 = 0 \\
&X_1, X_2, X_3, S_4, S_5, A_6, A_7 \geq 0.
\end{aligned} \tag{15}$$

Korak 2 – konstruiraj začetno tabelo. Vidimo, da je *m* (število pogojnih enačb) enak 4, *s* (število originalnih spremenljivk) je enak 3, *d* (število dopolnilnih, dodatnih in umetnih spremenljivk) pa je enak 4.

V razširjenem sistemu enačb nastopata dve dopolnilni in dve umetni spremenljivki. To so:

1. S_4 v prvi,
2. S_5 v drugi,
3. A_6 v tretji in
4. A_7 v četrti enačbi.

V tem zaporedju zasedejo mesta v začetnem *BVS*. Po vrsti zapišemo še koeficiente pogojnih enačb, v zadnji stolpec dodamo njihove vrednosti *RHS* in na konec dodamo vrstico z diferencami $Z_j - C_j$, ki jih izračunamo po obrazcu (4), in dobimo:

Tabela 10: Metoda simpleks, zgled 2, iteracija 1, korak 2

BVS	X_1	X_2	X_3	S_4	S_5	A_6	A_7	***RHS***
S_4	1	3	2	1	0	0	0	3
S_5	0	–2	1	0	1	0	0	1
A_6	1	–1	1	0	0	1	0	2
A_7	2	2	–3	0	0	0	1	0
$Z_j - C_j$	182	63	–126	0	0	0	0	

Korak 3 – poišči nov bazni vektor. Iz Tabele 10 razberemo, da največja diferenca $Z_j - C_j$ pripada spremenljivki X_1, zato je ta prvi kandidat za vstop v *BVS*.

Označimo $k = 1$.

Korak 4 – poišči vektor, ki bo izstopil. S pomočjo izraza (5) izračunamo koeficiente *CR*, pri čemer je $k = 1$:

Tabela 11: Metoda simpleks, zgled 2, iteracija 1, korak 4

BVS	X1	...	RHS	CRi,1
S4	1		3	3
S5	0		1	–
A6	1		2	2
A7	2		0	0

Najmanjši *CR* pripada četrti vrstici, zato bo izstopil vektor, ki je v *BVS* na četrtem mestu. To je A_7, indeks pivotne vrstice $r = 4$.

Korak 5 – generiraj novo tabelo s parametroma *k* in *r*. S pomočjo izrazov (6) in (7) izračunamo novo tabelo, ki se glasi:

Tabela 12: Metoda simpleks, zgled 2, iteracija 1, korak 5

BVS	X_1	X_2	X_3	S_4	S_5	A_6	A_7	***RHS***
S_4	0	2	3,5	1	0	0	–0,5	3
S_5	0	–2	1,0	0	1	0	0	1
A_6	0	–2	2,5	0	0	1	–0,5	2
X_1	1	1	–1,5	0	0	0	0,5	0
$Z_j - C_j$	0	–119	147,0	0	0	0	–91,0	

Korak 6 – preveri trenutno rešitev. Ker iz Tabele 12 razberemo, da obstajajo diference $Z_j - C_j$, ki so večje od nič, je nadaljnja izboljšava še možna. Zato ponovimo korak 3.

Korak 3 – poišči nov bazni vektor. Iz Tabele 12 razberemo, da največja diferenca $Z_j - C_j$ pripada spremenljivki X_3, zato je ta naslednji kandidat za vstop v *BVS*. Označimo $k = 3$.

Korak 4 – poišči vektor, ki bo izstopil. S pomočjo izraza (5) izračunamo koeficiente *CR*, pri čemer je $k = 3$:

Tabela 13: Metoda simpleks, zgled 2, iteracija 2, korak 4

BVS	...	X_3	...	***RHS***	$CR_{i,3}$
S_4		3,5		3	0,857
S_5		1,0		1	1,000
A_6		2,5		2	0,800
X_1		–1,5		0	–

Najmanjši *CR* pripada tretji vrstici, zato bo izstopil A_6.

Indeks pivotne vrstice $r = 3$.

Korak 5 – generiraj novo tabelo s parametroma *k* in *r*. S pomočjo izrazov (6) in (7) izračunamo novo tabelo, ki se glasi:

Tabela 14: Metoda simpleks, zgled 2, iteracija 2, korak 5

BVS	X_1	X_2	X_3	S_4	S_5	A_6	A_7	***RHS***
S_4	0	4,8	0	1	0	–1,4	0,2	0,2
S_5	0	–1,2	0	0	1	–0,4	0,2	0,2
X_3	0	–0,8	1	0	0	0,4	–0,2	0,8
X_1	1	–0,2	0	0	0	0,6	0,2	1,2
$Z_j - C_j$	0	–1,4	0	0	0	–58,8	–61,6	

Korak 6 – preveri trenutno rešitev. Ker iz Tabele 14 razberemo, da so vse diference $Z_j - C_j$ manjše ali enake nič, pomeni, da nadaljnja izboljšava ni možna.

Rešitev je ***optimalna***, vrednosti spremenljivk so:

$$X_1 = 1{,}2,\ X_2 = 0 \text{ in } X_3 = 0{,}8.$$

Ko to vstavimo v originalno namensko funkcijo, dobimo za maksimum vrednost ***–2,4***.

1.4 Računalniška implementacija algoritma

Algoritem je implementiran v modulu *ModuleSimplex.bas* programskega orodja SixPap. Implementacija je napisana v programskem jeziku Microsoft Visual Basic in upošteva načela objektno orientiranega programiranja.

Podatki se skozi iteracije algoritma vodijo v okviru objektnih stuktur, ki so podrobneje opisane v poglavju, ki govori o programskem orodju.

Postopek je implementiran z glavno proceduro, ki predstavlja kontrolni proces od kreiranja začetnih podatkovnih struktur do zaključka izvajanja algoritma s prikazom izračunanih rezultatov in njihovo kontrolo. Glavna procedura v skladu z vrednostmi kontrolnih točk kliče ustrezne podprocedure, ki predstavljajo korake oziroma faze algoritma.

V nadaljevanju sledi razlaga ključnih delov programske kode posameznih procedur. Izpis celotne kode modula je v Prilogi 2.

1.4.1 Nabor ključnih spremenljivk

Ključne spremenljivke, ki nastopajo v modulu, so:

1. *Problem* – krovni objekt problema, na katerem se izvaja algoritem,
2. *Definition* – podrejeni objekt problema, ki nosi podatke o definiciji problema,
3. *Result* – podrejeni objekt problema, ki nosi podatke o rezultatu izvajanja algoritma,
4. *Inc* – objekt, ki vodi različne števce za ugotavljanje učinkovitosti algoritma,
5. *minMax* – indikator vrste namenske funkcije: *MIN = –1 , MAX = 1*,
6. *M* – število pogojnih neenačb,
7. *S* – število originalnih spremenljivk namenske funkcije,
8. *D* – število dopolnilnih in dodatnih spremenljivk,
9. *U* – število umetnih spremenljivk,
10. *bigM* – vrednost velikega M,
11. *tableau*()– delovna tabela, ki služi za izvajanje pivotiranja,
12. *BVS()*– množica baznih vektorjev,
13. *k* – pivotni stolpec,
14. *r* – pivotna vrstica,
15. *stopExecuting* – indikator za zaključek izvajanja algoritma,
16. *existsZjPositive* – indikator obstoja pozitivne razlike *Zj – Cj*,
17. *biggestZjColumnIndex* – indeks stolpca z največjo vrednostjo *Zj – Cj*, ki predstavlja naslednji potencialni k,
18. *degeneracyOccured* – indikator degeneracije.

1.4.2 Kontrolna procedura algoritma

Kontrolna procedura se imenuje *ExecuteSimplex* in kot parameter sprejme objekt *Problem*, na katerem se bo izvajal algoritem standardne simpleks metode.

Predstavlja vstopno točko modula in vsebuje naslednje korake:

1. inicializacija algoritma,
2. konstruiranje začetne tabele,
3. iterativno iskanje optimalne rešitve,
4. izračun nove tabele,
5. preverjanje pojava degeneracije,
6. preverjanje rezultata.

Programsko kodo implementacije korakov prikazuje Slika 2.

Slika 2: Implementacija simpleks algoritma: vstopno – kontrolna procedura

```
Public Sub ExecuteSimplex(onProblem As ClassProblem)

    InitiateSimplex                                                    (1)

    ConstructInitialTableau                                            (2)

    Do While stopExecuting = False                                     (3)

        CalculateNewTableau                                            (4)

        If degeneracyOccured Then                                      (5)
            Result.Degeneracy = "YES"
        End If

    Loop

    Select Case Result.Status                                          (6)

        Case EnumStatus.Optimal
            ModuleGauss.CalculateOptimalSolution Result, Definition, tableau, BVS

        Case Else
            noteDetail.SingleLine "Status has changed: " & Result.StatusName
            noteDetail.SingleLine "Execution stopped.", , 1, 1, "="

    End Select

End Sub
```

1.4.3 Inicializacija algoritma

Pri inicializaciji algoritma se določi število dopolnilnih, dodatnih in umetnih spremenljivk, dimenzije in začetne vrednosti delovne tabele ter dimenzija in začetne vrednosti *BVS*. Inicializacija se izvede s klicem procedure *InitiateSimplex* in zajema naslednje korake (Slika 3):

1. nastavitev začetnih vrednosti splošnih spremenljivk, vrste namenske funkcije, števila njenih spremenljivk in števila pogojnih neenačb iz definicije problema,
2. izračun števila dopolnilnih, dodatnih in umetnih spremenljivk,
3. določitev dimenzij delovne tabele in polnjenje z začetnimi vrednostmi,
4. določitev dimenzije *BVS* in polnjenje z začetnimi vrednostmi,
5. izračun vrednosti velikega *M*,
6. določitev dimenzije *simplexCj* niza z upoštevanjem števila dodatnih, dopolnilnih in umetnih spremenljivk.

Slika 3: Implementacija simpleks algoritma: inicijalizacijska procedura

```
Private Sub InitiateSimplex()

    stopExecuting = False                                                    (1)
    existsZjPositive = False
    biggestZjColumnIndex = 0
    minMax = IIf(Definition.Objective = "MAX", -1, 1)
    M = Definition.M
    S = Definition.S

    D = 0                                                                    (2)
    U = 0
    For i = 1 To M

        D = IIf(Definition.ConstraintType(i) <> "=", D + 1, D)
        U = IIf(Definition.ConstraintType(i) <> "<", U + 1, U)

    Next i

    ReDim tableau(1 To M + 1, 1 To S + D + U + 1)                            (3)
    For i = 1 To M

        For j = 1 To S

            tableau(i, j) = Definition.A(i, j)

        Next j

    Next i

    ReDim BVS(1 To M)                                                        (4)

    bigM = 0                                                                 (5)
    For i = 1 To S
        bigM = IIf(Definition.Cj(i) > bigM, Definition.Cj(i), bigM)
    Next i
    bigM = 10 * bigM

    ReDim simplexCj(S + D + U + 1)                                           (6)
    For j = 1 To S

        simplexCj(j) = Definition.Cj(j):

    Next j

End Sub
```

1.4.4 Konstruiranje začetne tabele

Konstruiranje začetne tabele se izvede s klicem procedure *ConstructInitialTableau*.

Slika 4: Implementacija simpleks algoritma: procedura za konstruiranje začetne tabele

```
Private Sub ConstructInitialTableau()

    Dim Dj As Long: Dj = 0                                              (1)
    Dim Uj As Long: Uj = 0

    For i = 1 To M                                                      (2)
        Select Case Definition.ConstraintType(i)

        Case "="
            Uj = Uj + 1
            tableau(i, S + D + Uj) = 1
            simplexCj(S + D + Uj) = minMax * bigM
            BVS(i) = S + D + Uj

        Case "<"
            Dj = Dj + 1
            tableau(i, S + Dj) = 1
            BVS(i) = S + Dj

        Case ">"
            Uj = Uj + 1
            Dj = Dj + 1
            tableau(i, S + Dj) = -1
            simplexCj(S + D + Uj) = minMax * bigM
            tableau(i, S + D + Uj) = 1
            BVS(i) = S + D + Uj

        End Select

    Next i

    For i = 1 To M                                                      (3)
        tableau(i, S + D + U + 1) = Definition.RHS(i)
    Next i

    CalculateZ                                                          (4)

    stopExecuting = Not existsZjPositive                                (5)

End Sub
```

Procedura zajema naslednje korake (Slika 4):

1. določitev pomožnih spremenljivk,
2. kreiranje dopolnilnih, dodatnih in umetnih spremenljivk,
3. branje *RHS*,
4. izračun *Zj*,
5. kontrola obstoja pozitivnega *Zj*: če ne obstaja, prekini izvajanje.

1.4.5 Izračun nove tabele

Iskanje novega vstopnega in izstopnega vektorja ter izračun nove tabele se izvede s klicem procedure *CalculateNewTableau*. Procedura zajema naslednje korake (Slika 5):

1. iskanje najprimernejšega *k* in nastavitev začetnih vrednosti,
2. izračun novega *r* s pomočjo *CR* testa,
3. preverjanje rezultata *CR* testa: če *CR* test ne da rezultata, je rešitev neomejena,
4. pivotiranje tabele in preračun zadnje vrstice,
5. preveranje pozitivnega *Zj*: če pozitivni *Zj* ne obstaja, zaključi z izvajanjem.

Slika 5: Implementacija simpleks algoritma: procedura za izračun nove tabele

```
Private Sub CalculateNewTableau()

    k = biggestZjColumnIndex                                              (1)
    Result.Status = NotCalculated
    stopExecuting = True

    r = ModuleGauss.ColumnRatioTest(Inc, tableau, k, degeneracyOccured)   (2)

    If r = 0 Then                                                         (3)

        Result.Status = EnumStatus.Unbounded

    End If

    ModuleGauss.GaussPivot Inc, tableau, k, r, BVS                        (4)
    CalculateZ

    stopExecuting = Not existsZjPositive                                  (5)

End Sub
```

1.4.6 Preračun zadnje vrstice *Zj*

Zadnjo vrstico preračunamo s pomočjo funkcije *CalculateZ*.

Slika 6: Implementacija simpleks algoritma: funkcija za preračun zadnje vrstice Zj

```
Private Function CalculateZ()

    Dim tempZ As Double                                                          (1)
    Dim tempBiggestZj As Double: tempBiggestZj = 0
    existsZjPositive = False
    biggestZjColumnIndex = 0

    For j = 1 To S + D + U + 1                                                   (2)

        tempZ = 0
        For i = 1 To M

            tempZ = tempZ + simplexCj(BVS(i)) * tableau(i, j)                    (3)

        Next i

        If j < S + D + U + 1 Then

            tableau(M + 1, j) = minMax * (tempZ - simplexCj(j))                  (4)

            existsZjPositive = IIf(tableau(M + 1, j) > ModuleGauss.eta, True, existsZjPositive)   (5)

            If tableau(M + 1, j) > tempBiggestZj Then                            (6)

                tempBiggestZj = tableau(M + 1, j)
                biggestZjColumnIndex = j

            End If

        Else

            tableau(M + 1, j) = tempZ                                            (7)

        End If

    Next j

    If Not existsZjPositive Then Result.Status = Optimal                         (8)

End Function
```

Izračun poteka po naslednjih korakih (Slika 6):

1. nastavitev začetnih vrednosti začasnih spremenljivk,
2. preračun vrednosti zadnje vrstic,
3. izračun kumulativne vrednosti,
4. izračun vrednosti zadnje vrstice tabele,
5. preverjanje obstoja pozitivne vrednosti *Zj*,
6. ugotavljanje največje vrednosti *Zj*,
7. izračun trenutne vrednosti namenske funkcije,
8. preverjanje optimalnosti rešitve.

1.5 Računalniška implementacija Gaussovih transformacijskih pravil

Obe obravnavani metodi linearnega programiranja uporabljata Gaussova transformacijska pravila, zato so ta implementirana v svojem modulu *ModuleSimplex.bas* programskega orodja SixPap. Implementacija je napisana v programskem jeziku Microsoft Visual Basic in upošteva načela objektno orientiranega programiranja.

Modul zajema naslednje procedure in funkcije:

1. *GaussPivot* – pivotiranje tabele po Gausovi transformaciji,
2. *ColumnRatioTest* – iskanje pivotne vrstice s pomočjo količnika CR,
3. *RowRatioTest* – iskanje pivotne vrstice in stolpca s pomočjo količnika RR,
4. *CalculateOptimalSolution* – izračun optimalne rešitve na podlagi rezultata uporabljene metode.

V nadaljevanju je razlaga ključnih delov programske kode posameznih procedur. Izpis celotne kode modula je v Prilogi 3.

1.5.1 Pivotiranje tabele

Pravila za pivotiranje tabele so implementirana v proceduri *GaussPivot*, ki sprejme naslednje parametre:

1. *Inc* – objekt razreda *ClassCounters*, kjer se vodijo števci učinkovitosti,
2. *tableau* – delovna tabela,
3. *k* – pivotni stolpec, ki predstavlja nov bazni vektor v *BVS*,
4. *r* – pivotna vrstica, ki predstavlja bazni vektor, ki bo izstopil iz *BVS* in
5. *BVS* – množica baznih vektorjev.

Procedura se izvaja po naslednjih korakih (Slika 7):

1. določitev ključnega koeficienta za pivotiranje,
2. določitev novega baznega vektorja,
3. izračun koeficientov v izbrani vrstici *r*,
4. preračun koeficientov ostalih vrstic.

Slika 7: Implementacija pivotiranja tabele po Gaussovih transformacijskih pravilih

```
Public Sub GaussPivot(Inc, tableau,k, r, BVS)

    keyA = tableau(r, k)                                          (1)

    BVS(r) = k                                                    (2)

    For j = LBound(tableau, 2) To UBound(tableau, 2)              (3)

        tableau(r, j) = tableau(r, j) / keyA

    Next j


    For i = LBound(tableau, 1) To UBound(tableau, 1)              (4)

        If i = r Then i = i + 1

        For j = LBound(tableau, 2) To UBound(tableau, 2)

            If j = k Then j = j + 1

            tableau(i, j) = tableau(i, j) - tableau(i, k) * tableau(r, j)

        Next j

        tableau(i, k) = 0

    Next i

End Sub
```

1.5.2 Iskanje pivotne vrstice s pomočjo količnika *CR*

Iskanje baznega vektorja, ki bo izstopil (vrstica *r*) s pomočjo najmanjše pozitivne vrednosti količnika *CR*, je implementirano v funkciji *ColumnRatioTest*.

Slika 8: Implementacija iskanja pivotne vrstice s pomočjo količnika CR

```
Public Function ColumnRatioTest(Inc, tableau, k, degeneracyOccured, ignoreDegeneracy) As Long
    minimalColumnRatio = -1                                                              (1)
    potencialRs(1) = 0
    ColumnRatioTest = 0
    For i = 1 To UBound(tableau, 1) - 1                                                  (2)
        If ((tableau(i, k) > 0) And (tableau(i, UBound(tableau, 2)) >= 0)) Then
            tempColumnRatio = tableau(i, UBound(tableau, 2)) / tableau(i, k)
            If tempColumnRatio = minimalColumnRatio Then
                tempDegeneracy = True
                ReDim Preserve potencialRs(1 To UBound(potencialRs) + 1)
                potencialRs(UBound(potencialRs)) = i
            End If
            If ((tempColumnRatio < minimalColumnRatio) Or (minimalColumnRatio = -1)) Then
                tempDegeneracy = False
                ReDim potencialRs(1 To 1): potencialRs(1) = i
                minimalColumnRatio = tempColumnRatio
            End If
        End If
    Next i
    If ignoreDegeneracy Then                                                             (3)
        degeneracyOccured = False
        ColumnRatioTest = potencialRs(1)
        Exit Function
    End If
    If tempDegeneracy Then
        j = 0
        Do While tempR = 0
            j = j + 1
            If j = k Then j = j + 1
            If j > UBound(tableau, 2) - 1 Then Exit Do
            minimalColumnRatio = -1
            For i = LBound(potencialRs) To UBound(potencialRs)
                If ((tableau(potencialRs(i), k) > 0) And (tableau(potencialRs(i), j)) >= 0) Then
                    tempColumnRatio = tableau(potencialRs(i), j) / tableau(potencialRs(i), k)
                    If tempColumnRatio = minimalColumnRatio Then tempR = 0
                    If ((tempColumnRatio < minimalColumnRatio) Or (minimalColumnRatio = -1)) Then
                        tempR = potencialRs(i)
                        minimalColumnRatio = tempColumnRatio
                    End If
                End If
            Next i
        Loop
    Else tempR = potencialRs(1) End If
    degeneracyOccured = tempDegeneracy                                                   (4)
    ColumnRatioTest = tempR
End Function
```

Funkcija sprejme naslednje parametre:

1. *Inc* – objekt razreda *ClassCounters,* kjer se vodijo števci učinkovitosti,
2. *tableau* – delovna tabela,
3. *k* – pivotni stolpec,
4. *degeneracyOccured* – referenca na indikator pojava degeneracije in
5. *ignoreDegeneracy* – indikator načina obravnavanja degeneracije.

Izvajanje iskanja poteka po naslednjih korakih (Slika 8):

1. določitev začetnih vrednosti najmanjšega *CR,* nabora potencialnih *r*-jev in začetne vrednosti funkcije,
2. iterativno preverjanje količnikov *CR,* upoštevajo se samo pozitivne komponente novega vektorja,
3. preverjanje možnosti pojava degeneracije in ustrezno obravnavanje,
4. priprava rezultata.

1.5.3 Iskanje pivotne vrstice in stolpca s pomočjo količnika RR

Iskanje pivotne vrstice in stolpca s pomočjo pravil, ki veljajo za reševanje LP problema po simpleks metodi z uporabo dualnega programa in največje pozitivne vrednosti količnika *RR,* je implementirano v funkciji *ColumnRatioTest*.

Funkcija sprejme sprejme naslednje parametre:

1. *Inc* – objekt razreda *ClassCounters*, kjer se vodijo števci učinkovitosti,
2. *tableau* – delovna tabela,
3. *k* – referenca na pivotni stolpec in
4. *r* – referenca na pivotno vrstico.

Izvajanje iskanja poteka po naslednjih korakih (Slika 9):

1. inicializacija spremenljivk,
2. inicializacija rezultata,
3. priprava nabora potencialnih vrednosti *r* pri negativnih *RHS*,
4. če je najmanjši *RHS* enak novemu dodaj vrstico v nabor,
5. če je novi RHS manjši od prejšnjega najmanjšega postane vrstica edina vrednost v naboru,
6. iskanje najboljšega količnika *RR* pri negativnih *Cj* in *Aij*,
7. priprava rezultata.

Slika 9: Implementacija iskanja pivotne vrstice in stolpca z RR testom

```
Public Sub RowRatioTest(Inc, tableau, k, r)

    Dim minimalRHS As Double: minimalRHS = 0                                    (1)
    Dim maximalRowRatio As Double
    Dim tempRowRatio As Double
    Dim tempR() As Long

    k = 0                                                                       (2)
    r = 0

    For i = 1 To UBound(tableau, 1) - 1                                         (3)
        If tableau(i, UBound(tableau, 2)) < 0 Then

            If minimalRHS = tableau(i, UBound(tableau, 2)) Then                 (4)
                ReDim Preserve tempR(UBound(tempR) + 1)
                tempR(UBound(tempR)) = i
            End If

            If tableau(i, UBound(tableau, 2)) < minimalRHS Then                 (5)
                ReDim tempR(0)
                tempR(0) = i
                minimalRHS = tableau(i, UBound(tableau, 2))
            End If
        End If
    Next i

    maximalRowRatio = 0
    For i = LBound(tempR) To UBound(tempR)                                      (6)
        For j = 1 To UBound(tableau, 2) - 1
            If tableau(UBound(tableau, 1), j) < 0 Then
                If tableau(tempR(i), j) < 0 Then

                    tempRowRatio = tableau(UBound(tableau, 1), j) / tableau(tempR(i), j)

                    If maximalRowRatio < tempRowRatio Then                      (7)

                        maximalRowRatio = tempRowRatio
                        r = tempR(i)
                        k = j

                    End If
                End If
            End If
        Next j
    Next i
End Sub
```

Slika 10: Implementacija izračuna vrednosti optimalne rešitve

```
Public Sub CalculateOptimalSolution(Result, Definition, tableau, BVS)

    Dim trivialSolution As Boolean: trivialSolution = True

    For j = 1 To Definition.S                                                    (1)
        Result.Xj(j) = 0
    Next j
    Result.FunctionValue = 0
    For i = 1 To Definition.M
        Result.BVS(i) = BVS(i)
        If (BVS(i) > 0) And (BVS(i) <= Definition.S) Then
            Result.Xj(BVS(i)) = tableau(i, UBound(tableau, 2))
            trivialSolution = IIf(Result.Xj(BVS(i)) <> 0, False, trivialSolution)
            Result.FunctionValue = Result.FunctionValue + Result.Xj(BVS(i)) * Definition.Cj(BVS(i))
        End If
    Next i

    Result.TestResult = "FAILED"                                                 (2)
    Dim tempValue As Double
    Dim tempTest As Boolean: tempTest = True
    For i = 1 To Definition.M
        tempValue = 0
        For j = 1 To Definition.S
            tempValue = tempValue + Definition.A(i, j) * Result.Xj(j)
        Next j
        Result.ConstraintTest(i) = False
        Select Case Definition.ConstraintType(i)
            Case "=": If Abs(tempValue - Definition.RHS(i)) <= eta Then
                    Result.ConstraintTest(i) = True
                End If
            Case ">": If tempValue >= Definition.RHS(i) - eta Then
                    Result.ConstraintTest(i) = True
                End If
            Case "<": If tempValue <= Definition.RHS(i) + eta Then
                    Result.ConstraintTest(i) = True
                End If
        End Select
        tempTest = (tempTest And Result.ConstraintTest(i))
    Next i

    Result.TestResult = IIf(tempTest, "OK", "FAILED")                            (3)
    If trivialSolution Then Result.Status = Trivial
    If Result.TestResult = "FAILED" Then Result.Status = TestFailed

End Sub
```

1.5.4 Izračun vrednosti optimalne rešitve

Izračun vrednosti rešitve problema in testiranje na pogojne neenačbe sta implementirana v proceduri *CalculateOptimalSolution*, ki sprejme naslednje parametre:

1. *Result* – objekt razreda *ClassProblemResult*, kjer se vodijo podatki o rezultatu izvajanja posamezne metode,
2. *Definition* – objekt razreda *ClassProblemDefinition*, kjer se vodijo podatki o definiciji problema,
3. *tableau* – delovna tabela in
4. *BVS* – množica baznih vektorjev.

Izračun optimalne rešitve zajema naslednje korake (Slika 10):

1. izračun vrednosti namenske funkcije,
2. testiranje rešitve na pogojne neenačbe
3. ugotavljanje statusa rezultata

2 METODA POTISNI-POVLECI

Reševanje problemov linearnega programiranja, pri katerih nastopajo pogojne linearne neenačbe tipa ">=" (proizvodne omejitve) s pozitivno desno stranjo in pogojne linearne enačbe (tipa "="), povzroča preglavice, odkar obstaja linearno programiranje (Chvatal, 1983).

Ena od izvedb simpleks metode, imenovana dvofazna metoda, se problema loteva tako, da vpelje dodatno, umetno namensko funkcijo, ki predstavlja vsoto umetnih spremenljivk. Druga, popularno imenovana veliki M (angl. *Big M*), pa originalni enačbi doda vsoto umetnih spremenljivk, pomnoženih z zelo velikim pozitivnim številom (Ravindran, Ragsdell, & Reklaitis, 2006).

Tudi uporaba dualne simpleks metode prinaša svoje probleme, saj je treba v primeru, da kateri od koeficientov namenske funkcije ni dualno rešljiv, uvesti umetno pogojno linearno neenačbo, utrudljivo pa je tudi obravnavanje pogojnih enačb.

Leta 1997 Arsham predstavi novo metodo, ki jo poimenuje potisni-povleci. Algoritem temelji na dobro znani simpleks metodi, vendar za razliko od te pri reševanju problemov LP ne uporablja umetnih spremenljivk oziroma umetne funkcije pri problemih, ki v osnovni obliki niso rešljivi (Arsham, 1997).

Algoritem je sestavljen iz pripravljalne faze ter dveh glavnih faz: "potisni" (angl. *Push*) in "povleci" (angl. *Pull*). V pripravljalni fazi se dani problem z enostavnimi prijemi preoblikuje v ustrezno obliko.

Prva faza, "potisni", v prvem delu izoblikuje *BVS*, za katerega je možno, da dani problem rešitev ima ali pa ne. Za razliko od standardne simpleks in dualne simpleks metode je pri metodi potisni-povleci *BVS* na začetku nepoln, lahko celo prazen, z vsako naslednjo iteracijo pa pridobi po eno bazno spremenljivko.

Ko je *BVS* poln, pogoj za optimalno rešitev pa še ni izpolnjen, preide algoritem v drugi del faze "potisni", pri čemer možno rešitev na podoben način kot standardna simpleks metoda potiska proti optimalni rešitvi. Pri tem se lahko zgodi, da algoritem generira *BVS*, za katerega sistem ni rešljiv.

V tem primeru nastopi druga glavna faza, imenovana "povleci". V tej fazi algoritem možno rešitev povleče nazaj na področje rešljivosti s pomočjo pravil, ki se uporabljajo pri dualnem simpleksu.

Vse faze uporabljajo običajna Gaussova pravila za transformacijo vrstic in se uspešno zaključijo ali pa nakažejo na nerešljivost problema (angl. *not feasible*) oziroma na neomejeno množico rešitev (angl. *unbounded*).

V nadaljevanju je predstavljena teoretična podlaga metode potisni-povleci, ki je povzeta po Arshamu (1997).

2.1 Teoretična podlaga

Ena od prednosti na simpleksu temelječe metode pred ostalimi je, da so vsi podatki, ki so potrebni za izvajanje analize občutljivosti LP, že zajeti v končni tabeli, ki jo generira algoritem. Povrhu novi algoritem deluje v prostoru originalnih spremenljivk, kar bistveno olajša geometrično predstavo njegovih postopkov. Ta predstava je zanimiva zlasti v primerjavi z geometrijo, ki stoji za standardno simpleks metodo. Standardna simpleks metoda je metoda, ki išče ekstremne točke (vogalna točka konveksnega poliedra). Začetna točka iskanja je navadno zelo daleč od optimalne rešitve. Metoda se giblje po presečiščih mejnih hiperravnin pogojnih linearnih neenačb, preskakujoč s trenutne najdene ekstremne točke na sosednjo, dokler po dveh fazah ne doseže optimalne. Za uspešno izvedbo potrebuje umetne spremenljivke, saj se v začetni točki nahaja izven območja izvedljivosti rešitve. V prvi fazi, ki se začne v začetni točki, skače standardna simpleks metoda po ekstremnih točkah, dokler ne naleti na točko, ki leži znotraj območja izračunljivosti rešitve. Ko to doseže – kar pomeni, da so iz množice baznih vektorjev izločene vse umetne spremenljivke, se simpleks metoda premika po robu območja izračunljivosti in z izboljševanjem vrednosti namenske funkcije poskuša doseči ekstremno točko (vogal

konveksnega poliedra), ki predstavlja optimum namenske funkcije pri danih pogojih. To pomeni, da poskuša standardna simpleks metoda v prvi fazi doseči območje izračunljivosti rešitve, v drugi fazi pa teži k optimalni rešitvi.

Metoda potisni-povleci pa se reševanja loteva drugače. V prvem delu faze "potisni" poskuša napolniti *BVS*, ki predstavlja presečišče mejnih hiperravnin pogojnih neenačb – ekstremno točko konveksnega poliedra (vogal). Pripravljalna faza (inicializacija) zagotovi začetno pozicijo nekje na presečišču nekaj mejnih hiperravnin, kar se odraža v nepopolni startni množici baznih vektorjev (vsebuje prazne vrstice). Ideja algoritma je doseči področje izračunljivosti točno nekje na njegovem robu. V prvi fazi, algoritem že potiska trenutno rešitev proti optimalni ekstremni točki, medtem ko simpleks teži le k temu, da bo dosežena ekstremna točka znotraj področja izračunljivosti. Dodajanje nove spremenljivke v množico baznih vektorjev pomeni dejansko prihod na mejno hiperravnino pripadajoče pogojne neenačbe. Tako metoda potisni-povleci izboljšuje trenutno možno rešitev z dodajanjem naslednje mejne hiperravnine k trenutnemu presečišču. Z omejitvijo vstopajočih spremenljivk v množico baznih vektorjev na tiste, ki so še proste, zagotavlja faza "potisni" gibanje po prostoru presečišč hiperravnin, določenih v pripravljalni fazi, dokler ni najdena naslednja ustrezna hiperravnina. V tem delu algoritma ne prihaja do zamenjav v množici baznih vektorjev. Z vsako iteracijo se zmanjša število dimenzij aktualnega področja, dokler ni množica baznih vektorjev polna in predstavlja ekstremno točko. V tem delu prve faze ne more priti do degeneracije.

Izbor vstopne spremenljivke, ki ima največji *Cj*, pripomore k premiku proti optimalni ekstremni točki. Zato se drugi del faze "potisni" že začne z ekstremno točko, konča pa s polno bazo in ekstremno točko v bližini optimalne. Če se nahaja v področju izračunljive rešitve, je to istočasno tudi optimalna rešitev, če pa ne, je faza "potisni" potisnila predaleč. Za razliko od simpleksovih najdenih ekstremnih točk, ki niso v področju izračunljive rešitve, se ta nahaja na drugi strani optimalne ekstremne točke ("potisni" je šel predaleč). Podobno kot pri dualni simpleks metodi v tem primeru nastopi faza "povleci", ki se giblje od ene ekstremne točke k drugi, s ciljem poiskati tako točko, ki je znotraj področja izračunljivosti in zadošča pogojem optimalnosti. Tudi v tej fazi ne more priti do degeneracije, saj izloča elemente z negativnimi, od nič različnimi vrednostmi *RHS*.

Teorem 1. Z izvajanjem korakov 3.1 in 3.2 je vedno mogoče generirati polno množico baznih vektorjev, ki pa ne predstavlja nujno izračunljive rešitve.

Dokaz. Prvi del trditve je možno dokazati s pomočjo protislovja ob upoštevanju dejstva, da se pogojne neenačbe ne podvajajo. Drugi del, neizračunljiva rešitev, pa je posledica dejstva, da gremo pri potiskanju proti optimalni rešitvi lahko predaleč, tako da pade rešitev izven področja izračunljivosti.

Če so vsi koeficienti v kateri koli vrstici enaki nič, to pomeni dva specialna primera.

V prvem primeru je tudi desna stran enaka 0. Taka vrstica predstavlja redundandno pogojno neenačbo, zato jo enostavno izbrišemo in nadaljujemo z izvajanjem algoritma.

V drugem primeru, ko desna stran ni enaka nič, problem nima izračunljive rešitve.

Teorem 2. V prvem delu faze "potisni" in v fazi "povleci" ne more priti do degeneracije, ki bi lahko povzročila gibanje v neskončni zanki.

Dokaz. Znano je, da v primeru, ko je v kateri koli simpleks tabeli neka vrednost desnih strani enačb enaka nič (razen v zaključni tabeli), obstaja možnost pojava degeneracije v naslednji iteraciji. V prvem delu faze "potisni" ne more priti do degeneracije, saj ne zamenjavamo spremenljivk, temveč samo širimo *BVS* z dodajanjem novih spremenljivk na prazna mesta. Faza "povleci" pa uporablja običajna dualni-simpleks pravila za ugotavljanje izstopajoče spremenljivke. Tudi tu ne more priti do degeneracije, saj je cilj zamenjati vrstico z negativno, od nič različno vrednostjo *RHS*.

Za razliko od običajnih simpleks algoritmov zato pri obravnavanem algoritmu skoraj ne more priti do degeneracije. Degeneracija se lahko pojavi le v drugem delu prve faze, ko je *BVS* polna in nastopi uporaba običajnih simpleks pravil. V takem primeru je potrebno poklicati na pomoč pravila za obravnavanje degeneracije.

Teorem 3. Potisni-povleci algoritem vedno konča v končnem številu iteracij.

Dokaz. Ker med prvim in drugim delom faze "potisni" ter fazo "povleci" ni povratnih zank, je dovolj pokazati, da se vsak od teh delov algoritma uspešno zaključi. Predpriprava že na osnovi formulacije problema delno zapolni *BVS,* pri čemer ne uporablja Gaussovih transformacijskih pravil. Začetni del faze "potisni" poskuša zapolniti prazna mesta v *BVS*. Ker je ta omejena, se mora tudi ta del končati v končno mnogo korakih. Nadaljevanje faze potisni uporablja običajna simpleks pravila, faza povleci pa običajna dualni-simpleks pravila, za katera pa je že dokazano, da končajo v končno mnogo korakih.

2.2 Algoritem

V nadaljevanju je podrobnejša razlaga izvajanja algoritma, povzeta po Arshamu (1997).

2.2.1 Začetni pogoji – predpriprava

Za izvajanje algoritma potisni-povleci mora biti problem linearnega programiranja podan v naslednji standardni obliki:

1. Problem mora biti t. i. **maksimizacijski** problem (iskanje maksimuma namenske funkcije pri danih pogojih). Če je problem podan v minimizacijski obliki, celotno

namensko funkcijo pomnožimo s faktorjem –1. S tem pretvorimo problem v maksimizacijski. Optimalna rešitev se s tem ne spremeni, rešitev problema pa nato izračunamo s pomočjo originalne namenske funkcije.

2. Vse ***RHS*** morajo biti **nenegativne** (večje ali enake nič). V primeru negativnosti pomnožimo celotno pogojno neenačbo s faktorjem –1. Pri tem moramo biti pozorni na spremembo relacije v primeru neenakosti. Optimalna rešitev se s tem ne spremeni.
3. Vse **spremenljivke** morajo biti **nenegativne**.
   a) Če obstaja **neomejena** spremenljivka X_j (lahko je negativna, enaka nič ali pozitivna), lahko naredimo sledeče:
      - X_j nadomestimo (povsod, kjer nastopa) s parom $y - X_j$, kjer je y nenegativna spremenljivka. To poveča dimenzijo problema le za ena, ne glede na to, koliko spremenljivk je neomejenih (vedno uporabimo isti y).
      - Če obstaja med pogojnimi funkcijami kakšna enačba, lahko X_j izrazimo z ostalimi. S tem se znebimo ene spremenljivke in ene pogojne funkcije. Če ne obstaja neomejena spremenljivka, pa se pogojne enačbe vseeno poskušamo znebiti na tak način, lahko s tem povzročimo, da rešitev problema postane neizračunljiva.
   b) Če obstaja spremenljivka, ki je povsod **manjša od nič** X_j ; $X_j < 0$, jo v vseh enačbah nadomestimo z $-X_j$.

2.2.2 Faza "potisni" – korak 1: razširi sistem neenačb v sistem enačb

Z uvedbo dopolnilnih in dodatnih spremenljivk pretvori vse pogojne neenačbe (razen pogoja nenegativnosti) v enačbe. Postopek je enak kot pri standardni simpleks metodi, s to razliko, da tu ne uporabimo **umetnih** spremenljivk:

1. V primeru neenačbe tipa večje/enako (≥; t. i. produkcijska omejitev) vpeljemo **dodatno** spremenljivko z vrednostjo koeficienta –1, vrednost koeficienta namenske funkcije pri tej spremenljivki je enaka nič.
2. V primeru neenačbe tipa manjše/enako (≤; t. i. omejitev virov) vpeljemo **dopolnilno** spremenljivko z vrednostjo koeficienta +1, vrednost koeficienta namenske funkcije pri tej spremenljivki je enaka nič.

Pri vpeljavi dodatnih in dopolnilnih spremenljivk moramo biti pozorni na sledeče: matrika koeficientov pogojnih enačb mora imeti poln rang po vrsticah (rang matrike po vrsticah mora biti enak številu vseh vrstic) (Grasselli, 1986), sicer rešitev ne obstaja ali pa vsebuje redundantne vrstice.

V primeru, da so vsi elementi katere koli vrstice v kateri koli tabeli (velja tudi za tabele, ki predstavljajo delne rezultate oz. približke pri iskanju optimalne rešitve) enaki nič, to predstavlja enega izmed dveh posebnih primerov:

1. Če je element tabele, ki predstavlja *RHS*, različen od nič, je rešitev problema neizračunljiva.
2. Če je element tabele, ki predstavlja *RHS*, tudi enak nič, je ta vrstica redundantna. Tako vrstico brišemo iz tabele in nadaljujemo z izvajanjem algoritma.

2.2.3 Korak 2: konstruiraj začetno tabelo

Začetna tabela zajema matriko koeficientov pogojnih enačb, ki sta ji (na konec) dodana vrstica s koeficienti namenske funkcije in (tudi na konec) stolpec, ki vsebuje vrednosti desnih strani neenačb. V začetno množico *BVS* dodamo le dopolnilne spremenljivke.

Dimenzija *BVS* bo tako enaka *m* – številu pogojnih neenačb. Dimenzija tabele pa bo enaka *m + 1* vrstic in *s + d + 1* stolpcev, pri čemer pomeni *s* število originalnih, *d* število dodatnih in dopolnilnih spremenljivk, njuno vsoto pa označimo z *n*.

2.2.4 Korak 3: napolni množico baznih vektorjev

Rezultat tega koraka je polna množica baznih vektorjev, za katero pa ni nujno, da predstavlja izračunljivo rešitev. Polnjenje *BVS* poteka po naslednjih pravilih:

1. **pravilo:**
   a) **če** je v *BVS* samo še ena prazna vrstica, naj na njeno mesto **vstopi spremenljivka X_j**, pri kateri je C_j (koeficient namenske funkcije; vrednost elementa v *j*–tem stolpcu in zadnji vrstici tabele) **različen od nič** (C_j ima lahko tudi negativno vrednost) in ima med vsemi možnimi kandidati **najmanjšo nenegativno vrednost količnika *CR***. Izbrani stolpec (*j*) označimo s *k* (pivotni stolpec), edino prosto vrstico v *BVS* pa z *r*. Količnik *CR* izračunamo s pomočjo izraza (5), pri izračunu upoštevamo samo pozitivne komponente novega vektorja (stolpec *k*);
   b) **sicer** uporabi pravili 2a in 2b;
2. **pravilo:**
   a) v *BVS* naj **vstopi spremenljivka X_j**, pri kateri je C_j največji. Ustrezen C_j ima lahko tudi negativno vrednost, mora pa biti od nič različen. Izbrani stolpec (*j*) označimo s *k* (pivotni stolpec);
   b) poišči **pivotno vrstico**. Med vrsticami, ki še nimajo zasedenega elementa v *BVS*, poišči tisto, ki ima **najmanjšo nenegativno vrednost *CR***. Pri računanju količnikov *CR* uporabimo izraz (5). Če obstaja več vrstic z najmanjšo nenegativno vrednostjo *CR*, izberi katero koli. Izbrano vrstico označimo z *r*;

3. **pravilo:**
   a) **če** je ***BVS* zapolnjen** ali pa so bile **izrabljene vse možnosti X_j**, nadaljuj s **korakom 4**;

b) **sicer** generiraj novo tabelo s parametroma *k* in *r* ter ponovi **korak 3**. Novo tabelo generiramo s pomočjo transformacijskih pravil:
   - **koeficiente v izbrani vrstici** *r* delimo s ključnim koeficientom *A(r,k)* po enačbi (6),
   - **koeficiente v vseh drugih vrsticah** pa izračunamo z izrazom (7).

2.2.5 Korak 4: potisni dalje proti optimalni rešitvi

Če za vse C_j velja, da so **manjši/enaki nič** ($C_j \leq 0$), nadaljuj s **korakom 5.**

Sicer potiskaj dalje proti optimalni rešitvi s pomočjo naslednjih pravil:

1. V *BVS* naj **vstopi spremenljivka X_j,** pri kateri je C_j največja. Tokrat upoštevamo samo pozitivne vrednosti C_j. Izbrani stolpec (*j*) označimo s *k*.
2. Iz *BVS* naj ***izstopi*** tista spremenljivka, ki leži v vrstici, za katero ima ***CR* najmanjšo nenegativno vrednost**. Če obstaja več takih vrstic, izberi katero koli (ta pojav kaže na degeneracijo). Izbrano vrstico označimo z *r*. Količnik *CR* izračunamo s pomočjo že znanega izraza (5).

Če korak 4 spodleti, to lahko pomeni, da je rešitev problema neomejena. Da preprečimo morebitno napačno ugotovitev neomejenosti, uvedemo v tabelo novo omejitveno funkcijo oblike $\sum X_i + S = M$ s spremenljivko *S* na ustreznem mestu v *BVS*. V enačbi predstavlja *M* dovolj veliko poljubno število, X_i pa spremenljivke, pri katerih je v trenutni tabeli vrednost C_j pozitivna. Vhodna spremenljivka (stolpec *k*) je tista z največjim pozitivnim C_j, izhodna (vrstica *r*) pa nova spremenljivka *S*. Pri generiranju nove tabele bodo postale vse vrednosti C_j manjše/enake nič, zato lahko nadaljujemo s **korakom 5**.

2.2.6 Korak 5: preveri trenutno rešitev

1. Če velja, da so **vse vrednosti RHS nenegativne in vse vrednosti C_j manjše/enake nič** ($(\forall RHS \geq 0) \wedge (\forall C_j \leq 0)$), pomeni, da je rešitev **optimalna**. Če je število C_j z vrednostjo enako nič večje od števila elementov *BVS*, pomeni, da obstaja več alternativnih optimalnih rešitev. Ker se vrednost namenske funkcije ne spremeni, če v bazo uvedemo nebazno spremenljivko z vrednostjo $C_j = 0$, si tako spremenljivko izberemo za vhodno in s količnikom *CR* poiščemo še izhodno. Z generiranjem nove tabele dobimo alternativno optimalno rešitev (vrednost namenske funkcije ostane ista). Velja tudi, da so vse linearne kombinacije alternativnih optimalnih rešitev obenem tudi optimalne rešitve problema linearnega programiranja (Vadnal, 1986).
2. **Sicer** nadaljuj s **korakom 6 (faza "povleci")**.

2.2.7 Korak 6 – faza "povleci": povleci rešitev nazaj na področje izračunljivosti

1. S pomočjo pravil, ki veljajo za reševanje LP problema po simpleks metodi z uporabo dualnega programa, najprej poiščemo spremenljivko, ki jo bomo izločili iz *BVS*. To je spremenljivka z najmanjšo vrednostjo *RHS*. Izbrano vrstico označimo z *r*.
2. Nato poišči spremenljivko, ki bo vstopila v *BVS*, in sicer naj bo to tista, ki ima v izbrani pivotni vrstici *r* trenutne tabele negativen koeficient.
   Če obstaja več takih vrednosti, izberi tisto, ki ima največjo pozitivno vrednost *RR* (angl. *Row Ratio*). Izbrani stolpec je pivotni stolpec *k*. Količnik *RR* izračunamo po naslednjem obrazcu dualne simpleks metode:

$$RR(r, j) = \frac{C(j)}{A(r, j)}, \tag{16}$$

   pri čemer so:

   a) *RR(r, j)* – količnik *r*-te vrstice; iščemo največjega pozitivnega,
   b) *j* – stolpec, ki pripada spremenljivki *X(j)*,
   c) *r* – pivotna vrstica,
   d) *C(j)* – vrednost koeficienta namenske funkcije v trenutni tabeli,
   e) *A(r, j)* – element matrike koeficientov pogojnih neenačb z indeksoma *j* in *r*.

Če korak 6 spodleti, pomeni, da rešitev problema ni izračunljiva. V tem primeru **končaj** izvajanje algoritma, s**icer** generiraj novo tabelo in nadaljuj s **korakom 5**.

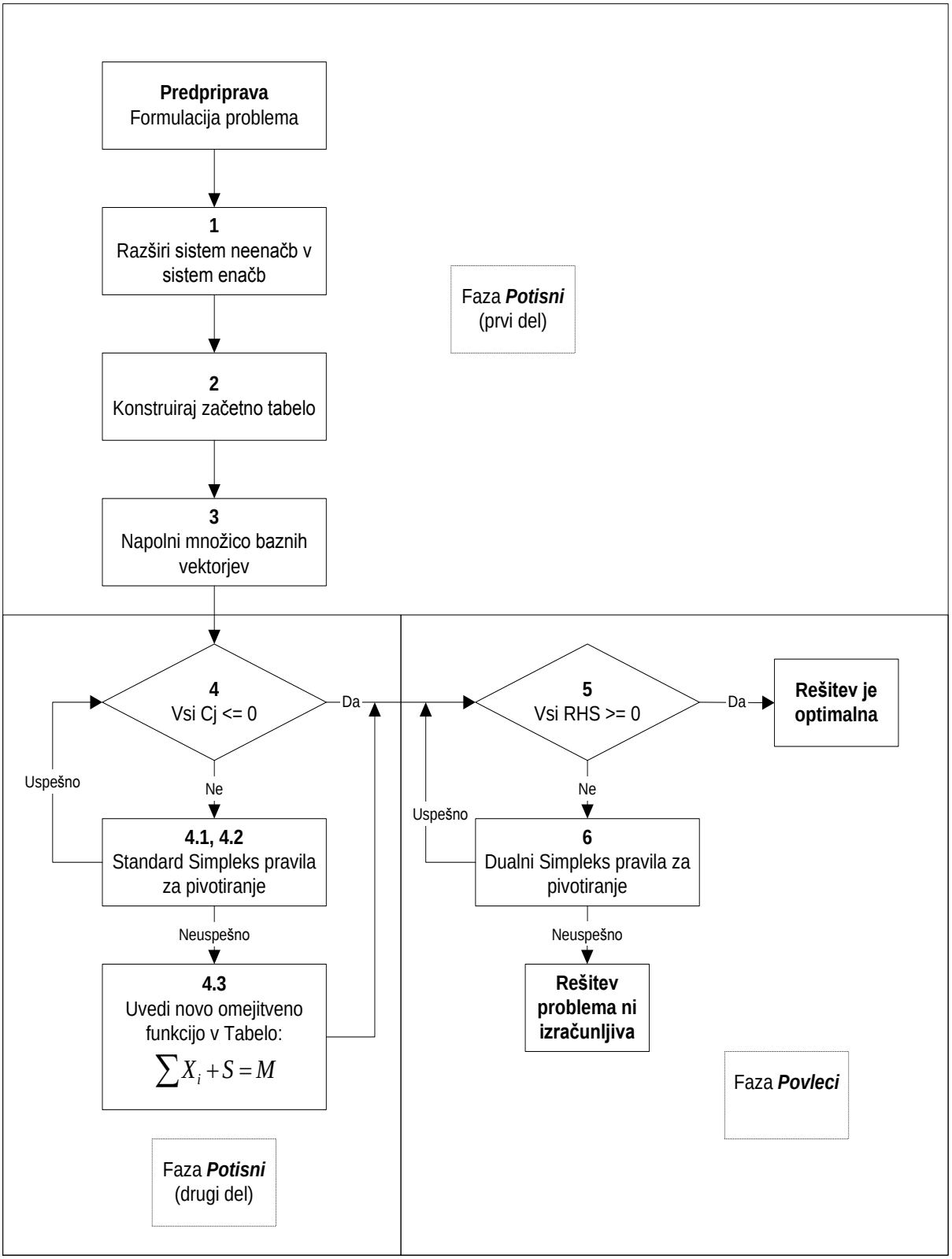
2.3 Diagram poteka algoritma

Potek izvajanja algoritma potisni-povleci je predstavljen z diagramom, ki ga prikazuje Slika 11.

Iz diagrama so jasno razvidni trije sklopi algoritma, ki sledijo en drugemu, brez povratnih zank. Ti sklopi so:

1. faza "potisni", prvi del, ki se začne s predpripravo in konča s polnim *BVS*,
2. faza "potisni", drugi del, ki *BVS*, za katerega se izkaže, da še ni optimalen, potiska dalje proti optimalni rešitvi in
3. faza "povleci", ki v primeru potiskanja predaleč, povleče rešitev nazaj na območje izračunljivosti.

Slika 11: Diagram poteka izvajanja algoritma po metodi potisni-povleci



Vir: Povzeto po spletni strani H. Arsham, The Push-and-Pull Solution Algorithm, 2016.

2.4 Primer reševanja problemov

V nadaljevanju je prikazana uporaba algoritma potisni-povleci pri iskanju optimalnih rešitev, in sicer na primeru istih dveh zgledov kot pri metodi simpleks.

2.4.1 Zgled 1

Iščemo maksimum namenske funkcije:

$$2X_1 + 6X_2 + 8X_3 + 5X_4, \tag{17}$$

ki zadošča naslednjim pogojem:

$$\begin{aligned} &4X_1 + X_2 + 2X_3 + 2X_4 \geq 80 \\ &2X_1 + 5X_2 + 4X_4 \leq 40 \\ &2X_2 + 4X_3 + X_4 = 120 \\ &X_1, X_2, X_3, X_4 \geq 0. \end{aligned} \tag{18}$$

Predpriprava. Na prvi pogled vidimo, da formulacija problema zadošča začetnim pogojem za izvajanje algoritma potisni-povleci, saj gre za maksimizacijski problem, vse desne strani pogojnih neenačb so nenegativne, isto velja za vse spremenljivke. Zato lahko nadaljujemo s prvim korakom algoritma.

Korak 1 – razširi sistem neenačb v sistem enačb. Prva neenačba predstavlja t. i. produkcijsko omejitev (tipa večje/enako), zato jo dopolnimo z dodatno spremenljivko S_5 s koeficientom –1.

Druga je tipa manjše/enako (predstavlja omejitev virov), zato jo dopolnimo z dopolnilno spremenljivko S_6 s koeficientom +1.

Tretje ni potrebno dopolnjevati (je že enačba).

Razširjeni sistem se tako glasi:

$$\begin{aligned} &\max: 2X_1 + 6X_2 + 8X_3 + 5X_4 \\ &4X_1 + X_2 + 2X_3 + 2X_4 - S_5 = 80 \\ &2X_1 + 5X_2 + 4X_4 + S_6 = 40 \\ &2X_2 + 4X_3 + X_4 = 120 \\ &X_1, X_2, X_3, X_4, S_5, S_6 \geq 0. \end{aligned} \tag{19}$$

Korak 2 – konstruiraj začetno tabelo. Vidimo, da je *m* (število pogojnih enačb) enak 3, *s* (število originalnih spremenljivk) je enak 4, *d* (število dopolnilnih in dodatnih spremenljivk) pa je enak 2.

V razširjenem sistemu enačb nastopa ena sama dopolnilna spremenljivka. To je spremenljivka S_6, nastopa pa v drugi pogojni enačbi, zato bo v začetnem *BVS* zasedla drugo mesto, medtem ko ostaneta prvo in tretje mesto prazna. Zapišemo po vrsti še koeficiente pogojnih enačb, v zadnji stolpec dodamo njihove vrednosti na desni strani *RHS* in na konec dodamo vrstico s koeficienti namenske funkcije C_j ter dobimo naslednjo tabelo:

Tabela 15: Metoda potisni-povleci, zgled 1, iteracija 1, korak 2

BVS	X_1	X_2	X_3	X_4	S_5	S_6	***RHS***
	4	1	2	2	–1	0	80
S_6	2	5	0	4	0	1	40
	0	2	4	1	0	0	120
C_j	2	6	8	5	0	0	

Korak 3 – napolni *BVS*. Po pravilu 1 koraka 3 vidimo, da v *BVS* obstaja več kot ena prazna vrstica. Z uporabo pravila 2a koraka 3 ugotovimo, da je C_j pri X_3 največji, zato ta postane prvi kandidat za vstop v *BVS*. S pomočjo izraza (5) izračunamo koeficiente *CR*, pri čemer je $k = 3$:

Tabela 16: Metoda potisni-povleci, zgled 1, iteracija 1, korak 3, pravilo 1

BVS	...	X_3	...	***RHS***	$CR_{i,3}$
		2		80	40,00
S_6		0		40	–
		4		120	30,00

Po pravilu 2b koraka 3 ugotovimo, da ima najmanjši *CR*, ki pripada prazni vrstici v *BVS*, tretja vrstica. Zato je $r=3$ in v *BVS* vstopi X_3, in sicer na tretje mesto. S pomočjo izrazov (6) in (7) izračunamo novo tabelo, ki se glasi:

Tabela 17: Metoda potisni-povleci, zgled 1, iretacija 1, korak 3, pravilo 2b

BVS	X_1	X_2	X_3	X_4	S_5	S_6	***RHS***
	4	0	0	1,50	–1	0	20
S_6	2	5,0	0	4,00	0	1	40
X_3	0	0,5	1	0,25	0	0	30
C_j	2	2,0	0	3,00	0	0	

Po pravilu 3 koraka 3 ugotovimo, da *BVS* še ni poln, zato ponovno uporabimo pravilo 1 koraka 3.

Tabela 18: Metoda potisni-povleci, zgled 1, iteracija 2, korak 3, pravilo 1

BVS	X_1	X_2	...	X_4	...	***RHS***	$CR_{i,1}$	$CR_{i,2}$	$CR_{i,4}$
	4	0,0		1,50		20	5,0	–	13,3
S_6	2	5,0		4,00		40	20,0	8,0	10,0
X_3	0	0,5		0,25		30	–	60,0	120,0

Izračun koeficientov *CR* za prvo vrstico (zadnjo prazno) in ustrezne kandidate (X_1, X_2 in X_4) nam pokaže, da je količnik najmanjši pri spremenljivki X_1. Zato postane $k = 1$ in $r = 1$.

S pomočjo izrazov (6) in (7) izračunamo novo tabelo, ki se glasi:

Tabela 19: metoda potisni-povleci, zgled 1, iteracija 2, korak 3, pravilo 2b

BVS	X_1	X_2	X_3	X_4	S_5	S_6	***RHS***
X_1	1	0,0	0	0,38	–0,25	0	5
S_6	0	5,0	0	3,25	0,50	1	30
X_3	0	0,5	1	0,25	0,00	0	30
C_j	0	2,0	0	2,25	0,50	0	

Po pravilu 3 koraka 3 ugotovimo, da je *BVS* poln in nadaljujemo s korakom 4.

Korak 4 – potisni dalje proti optimalni rešitvi. Ker v tabeli obstajajo C_j, ki so večji od nič, rešitev še ni optimalna.

Z uporabo pravila 1 koraka 4 ugotovimo, da je C_j pri X_4 največji, zato postane $k = 4$.

S pomočjo izraza (5) izračunamo pripadajoče *CR*:

Tabela 20: Metoda potisni-povleci, zgled 1, iteracija 2, korak 4, pravilo 1

BVS	...	X_4	...	***RHS***	$CR_{i,4}$
X_1		0,38		5	13,3
S_6		3,25		30	9,2
X_3		0,25		30	120,0

Po pravilu 2 koraka 4 ugotovimo, da je $r = 2$. Zato spremenljivka X_4 zamenja spremenljivko, ki je v *BVS* na drugem mestu, torej S_6.

S pomočjo izrazov (6) in (7) izračunamo novo tabelo, ki se glasi:

Tabela 21: Metoda potisni-povleci, zgled 1, iteracija 2, korak 4, pravilo 2

BVS	X_1	X_2	X_3	X_4	S_5	S_6	***RHS***
X_1	1	–0,58	0	0	–0,31	–0,12	1,54
X_4	0	1,54	0	1	0,15	0,31	9,23
X_3	0	0,12	1	0	–0,04	–0,08	27,70
C_j	0	–1,46	0	0	0,15	–0,69	

Ker v tabeli še vedno obstajajo C_j, ki so večji od nič, rešitev še ni optimalna.

Z uporabo pravila 1 koraka 4 ugotovimo, da je C_j pri S_5 največji, zato postane $k = 5$. S pomočjo izraza (5) izračunamo pripadajoče *CR*:

Tabela 22: Metoda potisni-povleci, zgled 1, iteracija 3, korak 4, pravilo 1

BVS	**...**	S_5	**...**	***RHS***	$CR_{i,5}$
X_1		–0,31		1,54	–
X_4		0,15		9,23	60,0
X_3		–0,04		27,70	–

Po pravilu 2 koraka 4 ugotovimo, da je $r = 2$. Zato spremenljivka S_5 zamenja spremenljivko, ki je v *BVS* na drugem mestu, torej X_4. S pomočjo izrazov (6) in (7) izračunamo novo tabelo, ki se glasi:

Tabela 23: Metoda potisni-povleci, zgled 1, iteracija 3, korak 4, pravilo 2

BVS	X_1	X_2	X_3	X_4	S_5	S_6	***RHS***
X_1	1	2,5	0	2,00	0	0,5	20
S_5	0	10,0	0	6,50	1	2,0	60
X_3	0	0,5	1	0,25	0	0,0	30
C_j	0	–3,0	0	–1,00	0	–1,0	

V tabeli ne obstaja C_j, ki bi bil večji od nič, zato nadaljujemo s **korakom 5**.

Korak 5 – preveri trenutno rešitev. V tabeli so vsi *RHS* večji od nič in vse C_j manjše/enake nič. Potrebe za uporabo faze "povleci" ni, rešitev je **optimalna**.

Vrednosti spremenljivk pri optimalni rešitvi so:

$$\boldsymbol{X_1 = 20, X_2 = 0, X_3 = 30} \text{ in } \boldsymbol{X_4 = 0}.$$

Ko to vstavimo v originalno namensko funkcijo, za maksimum dobimo vrednost ***280***.

2.4.2 Zgled 2

Iščemo maksimum namenske funkcije:

$$2X_1 + 3X_2 - 6X_3, \tag{20}$$

ki zadošča naslednjim pogojem:

$$\begin{aligned} &X_1 + 3X_2 + 2X_3 \le 3 \\ &-2X_2 + X_3 \le 1 \\ &X_1 - X_2 + X_3 = 2 \\ &2X_1 + 2X_2 - 3X_3 = 0 \\ &X_1, X_2, X_3 \ge 0. \end{aligned} \tag{21}$$

Predpriprava. Na prvi pogled vidimo, da formulacija problema zadošča začetnim pogojem za izvajanje algoritma *potisni-povleci*, saj gre za maksimizacijski problem, vse desne strani pogojnih neenačb so nenegativne, isto velja za vse spremenljivke. Zato lahko nadaljujemo s prvim korakom algoritma.

Korak 1 – razširi sistem neenačb v sistem enačb. Prvi dve neenačbi sta tipa manjše/enako (predstavlja omejitev virov), zato ju dopolnimo: prvo z dopolnilno spremenljivko S_4, drugo z dopolnilno spremenljivko S_5, obe s koeficientom +1.

Tretje in četrte ni treba dopolnjevati (je že enačba). Razširjeni sistem se tako glasi:

$$\begin{aligned} &\max : 2X_1 + 3X_2 - 6X_3 \\ &X_1 + 3X_2 + 2X_3 + S_4 = 3 \\ &-2X_2 + X_3 + S_5 = 1 \\ &X_1 - X_2 + X_3 = 2 \\ &2X_1 + 2X_2 - 3X_3 = 0 \\ &X_1, X_2, X_3, S_4, S_5 \ge 0. \end{aligned} \tag{23}$$

Korak 2 – konstruiraj začetno tabelo. Vidimo, da je $m = 4$, $s = 3$, $d = 2$ in $n = s + d = 5$. V razširjenem sistemu enačb nastopata dve dopolnilni spremenljivki, ki v začetnem *BVS* zasedeta prvo in drugo mesto, medtem ko ostaneta tretje in četrto mesto prazna.

Zapišemo po vrsti še koeficiente pogojnih enačb, v zadnji stolpec dodamo njihove vrednosti na desni strani (*RHS*) in na konec dodamo vrstico s koeficienti namenske funkcije C_j in dobimo naslednjo tabelo:

Tabela 24: Metoda potisni-povleci, zgled 2, iteracija 1, korak 2

BVS	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{S_4}$	$\mathbf{S_5}$	***RHS***
S_4	1	3	2	1	0	3
S_5	0	–2	1	0	1	1
	1	–1	1	0	0	2
	2	2	–3	0	0	0
C_j	2	3	–6	0	0	

Korak 3 – napolni *BVS*. Ker v *BVS* obstaja več kot ena prazna vrstica, po pravilu 2a koraka 3 ugotovimo, da je C_j pri X_2 največji, zato ta postane prvi kandidat za vstop v *BVS*. S pomočjo izraza (5) izračunamo koeficiente *CR*, pri čemer je $k = 2$:

Tabela 25: Metoda potisni-povleci, zgled 2, iteracija 1, korak 3, pravilo 2a

BVS	**...**	$\mathbf{X_2}$	**...**	***RHS***	$\mathbf{CR_{i,2}}$
S_4		3		3	1
S_5		–2		1	–
		–1		2	–
		2		0	0

Po pravilu 2b koraka 3 ugotovimo, da ima najmanjši ustrezen *CR*, ki pripada prazni vrstici v *BVS*, četrta vrstica. Zato je $r = 4$ in v *BVS* vstopi X_2, in sicer na četrto mesto. S pomočjo izrazov (6) in (7) izračunamo novo tabelo, ki se glasi:

Tabela 26: Metoda potisni-povleci, zgled 2, iteracija 1, korak 3, pravilo 2b

BVS	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{S_4}$	$\mathbf{S_5}$	***RHS***
S_4	–2	0	6,5	1	0	3
S_5	2	0	–2,0	0	1	1
	2	0	–0,5	0	0	2
X_2	1	1	–1,5	0	0	0
C_j	–1	0	–1,5	0	0	

Po pravilu 3 koraka 3 ugotovimo, da *BVS* še ni poln, zato ponovimo **korak 3**.

Tabela 27: metoda potisni-povleci, zgled 2, iteracija 2, korak 3, pravilo 1

BVS	$\mathbf{X_1}$	**...**	$\mathbf{X_3}$	**...**	***RHS***	$\mathbf{CR_{i,1}}$	$\mathbf{CR_{i,3}}$
S_4	–2		6,5		3	–	0,46
S_5	2		–2,0		1	0,5	–
	2		–0,5		2	1,0	–
X_2	1		–1,5		0	0,0	–

Po pravilu 1 koraka 3 vidimo, da je *CR* pri X_1 najmanjši (in edini ustrezen), zato ta postane naslednji kandidat za vstop v *BVS*. Glede na to, da je edino prosto mesto v *BVS* tretje, postane $r = 3$ in $k = 1$.

S pomočjo izrazov (6) in (7) izračunamo novo tabelo, ki se glasi:

Tabela 28: Metoda potisni-povleci, zgled 2, iteracija 2, korak 3, pravilo 3

BVS	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{S_4}$	$\mathbf{S_5}$	***RHS***
S_4	0	0	6,00	1	0	5
S_5	0	0	–1,50	0	1	–1
X_1	1	0	–0,25	0	0	1
X_2	0	1	–1,25	0	0	–1
C_j	0	0	–1,75	0	0	

Po pravilu 3 koraka 3 ugotovimo, da je *BVS* poln, zato nadaljujemo s **korakom 4**.

Korak 4 – potisni dalje proti optimalni rešitvi. Glede na to, da so vsi C_j manjši/enaki nič, nadaljujemo s **korakom 5**.

Korak 5 – preveri trenutno rešitev. Iz tabele razberemo, da obstajajo vrednosti *RHS*, ki so manjše od nič, zato nadaljujemo s **korakom 6**.

Korak 6 – povleci rešitev nazaj na področje izračunljivosti. Po pravilu 1 koraka 6 izberemo spremenljivko, ki bo izstopila iz *BVS*. To je spremenljivka, ki ima najmanjši vrednosti *RHS*, to sta $r = 2$ in $r = 4$.

S pomočjo izraza (16) izračunamo koeficiente *RR* za $r = 2$ in $r = 4$:

Tabela 29: Metoda potisni-povleci, zgled 2, iteracija 2, korak 6, pravilo 1

BVS	$\mathbf{X_1}$	$\mathbf{X_2}$	$\mathbf{X_3}$	$\mathbf{S_4}$	$\mathbf{S_5}$	***RHS***
S_4	0	0	6,0000	1	0	5
S_5	0	0	–1,5000	0	1	–1
X_1	1	0	–0,2500	0	0	1
X_2	0	1	–1,2500	0	0	–1
C_j	0	0	–1,7500	0	0	
$RR_{2,j}$	–	–	1,1667	–	0	
$RR_{4,j}$	–	0	1,4000	–	–	

Po pravilu 2 koraka 6 ugotovimo, da je $RR_{4,3}$ največji, zato postane $r = 4$ in $k = 3$.

S pomočjo izrazov (6) in (7) izračunamo novo tabelo, tabelo 30.

Tabela 30: Metoda potisni-povleci, zgled 2, iteracija 2, korak 6, pravilo 2

BVS	X_1	X_2	X_3	S_4	S_5	***RHS***
S_4	0	4,8	0	1	0	0,2
S_5	0	–1,2	0	0	1	0,2
X_1	1	–0,2	0	0	0	1,2
X_3	0	–0,8	1	0	0	0,8
C_j	0	–1,4	0	0	0	

Nadaljujemo s **korakom 5**.

Korak 5 – preveri trenutno rešitev. V tabeli so vsi *RHS* večji od nič in vse C_j manjše/enake nič, zato je rešitev **optimalna**. Vrednosti spremenljivk pri optimalni rešitvi so:

$$\boldsymbol{X_1 = 1,2}, \boldsymbol{X_2 = 0}, \text{ in } \boldsymbol{X_3 = 0,8}.$$

Ko to vstavimo v originalno namensko funkcijo, za maksimum dobimo vrednost ***–2,4***.

2.5 Računalniška implementacija algoritma

Algoritem je implementiran v modulu *ModulePushPull.bas* programskega orodja SixPap. Implementacija je napisana v programskem jeziku MS Visual Basic in upošteva načela objektno orientiranega programiranja.

Podatki se skozi iteracije algoritma vodijo v okviru objektnih stuktur, podrobneje opisanih v poglavju, ki govori o programskem orodju.

Postopek je implementiran z glavno proceduro, ki predstavlja kontrolni proces od kreiranja začetnih podatkovnih struktur do zaključka izvajanja algoritma s prikazom izračunanih rezultatov in njihovo kontrolo. Glavna procedura v skladu z vrednostmi kontrolnih točk kliče ustrezne podprocedure, ki predstavljajo korake oziroma faze algoritma.

V nadaljevanju sledi razlaga ključnih delov programske kode posameznih procedur. Izpis celotne kode modula je v Prilogi 1.

2.5.1 Nabor ključnih spremenljivk

Ključne spremenljivke, ki nastopajo v modulu, so:

1. *Problem* – krovni objekt problema, na katerem se izvaja algoritem,
2. *Definition* – podrejeni objekt problema, ki nosi podatke o definiciji problema,

3. *Result* – podrejeni objekt problema, ki nosi podatke o rezultatu izvajanja algoritma,
4. *Inc* – objekt, ki vodi različne števce za ugotavljanje učinkovitosti algoritma,
5. *minMax* – indikator vrste namenske funkcije: *MIN = –1 , MAX = 1*,
6. *M* – število pogojnih neenačb,
7. *S* – število originalnih spremenljivk namenske funkcije,
8. *D* – število dopolnilnih in dodatnih spremenljivk,
9. *U* – število umetnih spremenljivk,
10. *tableau*()– delovna tabela, ki služi za izvajanje pivotiranja,
11. *BVS()*– množica baznih vektorjev,
12. *k* – pivotni stolpec,
13. *r* – pivotna vrstica,
14. *stepName* – naziv koraka algoritma,
15. *sortedCj* – po velikosti urejen seznam vrednosti Cj,
16. *degeneracyOccured* – indikator degeneracije.

2.5.2 Kontrolna procedura algoritma

Kontrolna procedura se imenuje *ExecutePushPull* in kot parameter sprejme objekt *Problem*, na katerem se bo izvajal algoritem metode potisni-povleci.

Predstavlja vstopno točko modula in se izvaja po naslednjih korakih (Slika 12):

1. inicializacija algoritma,
2. ustrezna formulacija problema,
3. korak 1: razširi sistem neenačb v sistem enačb,
4. korak 2: konstruiraj začetno tabelo,
5. iterativno izvajanje koraka 3 s kontrolami,
6. korak 3: polnjenje *BVS*,
7. preverjanje možnost ponovitve koraka 3: če novi (k, r) ni bil najden, vse možnosti pa so bile izčrpane, zaključi korak 3 in nadaljuj s korakom 4,
8. preverjanje pojava degeneracije,
9. začetek drugega dela izvajanja algoritma s korakom 4,
10. korak 4: potisni dalje proti optimalni rešitvi,
11. iterativno izvajanje koraka 4 dokler obstajajo pozitivni *Cj*,
12. preverjanje pojava degeneracije,
13. korak 5: testiranje optimalnosti trenutne rešitve,
14. korak 6: povleci rešitev nazaj na področje izračunljivosti,
15. preverjanje pojava degeneracije,
16. rešitev je optimalna, zaključek izvajanja iteracij in
17. izračun in povratno preverjanje optimalne rešitve.

Slika 12: Implementacija potisni-povleci: vstopna in kontrolna procedura

```
Public Sub ExecutePushPull(onProblem As ClassProblem)
    InitiatePushPull                                                    (1)
    DoPreliminaries                                                     (2)
    DoStep1                                                             (3)
    DoStep2                                                             (4)
    Do While Not CompleteBVS                                            (5)
        sortedCj.RebuildWith tableau
        DoStep3                                                         (6)
        If sortedCj.IsEmpty Then                                        (7)
            noteDetail.SingleLine "All possible incoming Xj have been used."
            Exit Do
        End If
        If degeneracyOccured Then                                       (8)
            noteDetail.SingleLine "DEGENERACY OCCURED."
            Result.Degeneracy = "YES"
        End If
    Loop
    stepName = "4-PUSH"                                                 (9)
    Do While True
        Select Case stepName
        Case "4-PUSH"                                                   (10)
            sortedCj.RebuildWith tableau
            Do While sortedCj.Positive                                  (11)
                DoStep4
                If degeneracyOccured Then                               (12)
                    Result.Degeneracy = "YES"
                End If
                sortedCj.RebuildWith tableau
            Loop
            stepName = "5-TEST"
        Case "5-TEST": DoStep5()                                        (13)
        Case "6-PULL": DoStep6                                          (14)
            If degeneracyOccured Then                                   (15)
                Result.Degeneracy = "YES"
            End If
            stepName = "5-TEST"
        Case "OPTIMAL"                                                  (16)
            Result.Status = Optimal
            Exit Do
        End Select
    Loop

    ModuleGauss.CalculateOptimalSolution Result, Definition, tableau, BVS   (17)

    Exit Sub
End Sub
```

2.5.3 Inicializacija algoritma

Pri inicializaciji algoritma se določi število dopolnilnih in dodatnih spremenljivk, dimenzije in začetne vrednosti delovne tabele ter dimenzije in začetne vrednosti *BVS*.

Izvede se s klicem procedure *InitiatePushPull* po naslednjih korakih (
Slika *13*):

1. začetna nastavitev spremenljivk,
2. določitev vrste namenske funkcije, števila pogojnih neenačb in števila originalnih spremenljivk na podlagi definicije problema,
3. izračun števila dopolnilnih in dodatnih spremenljivk, število umetnih spremenljivk je nič,
4. določitev dimenzij delovne tabele in branje začetnih vrednosti iz definicije in
5. določitev dimenzij *BVS*.

Slika 13: Implementacija potisni-povleci: inicializacija algoritma

```
Private Sub InitiatePushPull()

    stopExecuting = False                                              (1)
    Result.Degeneracy = "NO"

    minMax = IIf(Definition.Objective = "MAX", -1, 1)                  (2)
    M = Definition.M
    S = Definition.S

    D = 0                                                              (3)
    U = 0
    For i = 1 To M
        D = IIf(Definition.ConstraintType(i) <> "=", D + 1, D)
    Next i

    ReDim tableau(1 To M + 1, 1 To S + D + 1)                          (4)
    For i = 1 To M
        For j = 1 To S
            tableau(i, j) = Definition.A(i, j)
        Next j
    Next i

    ReDim BVS(1 To M)                                                  (5)
End Sub
```

2.5.4 Ustrezna formulacija problema

Za izvajanje algoritma potisni-povleci mora biti problem maksimizacijski, vse desne strani neenačb morajo biti nenegativne, isto velja za vse spremenljivke.

Ustrezno preoblikovanje se izvede s klicem procedure *DoPreliminaries* po naslednjih korakih (Slika 14):

1. pretvorba v maksimizacijski problem in
2. ureditev pogojnih neenačb, ki imajo desno stran pozitivno.

Slika 14: Implementacija potisni-povleci: formulacija problema

```
Private Sub DoPreliminaries()

    For j = 1 To S                                                    (1)
        tableau(M + 1, j) = -1 * minMax * Definition.Cj(j)
    Next j

    For i = 1 To M                                                    (2)

        tableau(i, S + D + 1) = Abs(Definition.RHS(i))

        If Definition.RHS(i) < 0 Then
            For j = 1 To S
                tableau(i, j) = -1 * tableau(i, j)
            Next j
        End If

    Next i

End Sub
```

2.5.5 Korak 1: razširi sistem neenačb v sistem enačb

Razširitev sistema neenačb v sistem enačb se izvede s klicem procedure *DoStep1*.

Procedura implementira naslednja pravila (Slika 15):

1. pogojne enačbe ne generirajo dodatnih spremenljivk,
2. v primeru relacije ≤ vpeljava dopolnilne spremenljivke s koeficientom +1 in
3. v primeru relacije ≥ vpeljava dodatne spremenljivke s koeficientom -1.

Slika 15: Implementacija potisni-povleci: razširitev sistema neenačb v sistem enačb

```
Private Sub DoStep1()
    j = S
    For i = 1 To M                                                          (1)
        If Definition.ConstraintType(i) <> "=" Then
            j = j + 1
            tableau(i, j) = IIf(Definition.ConstraintType(i) = "<", 1, -1)  (2, 3)
            If (Definition.RHS(i) < 0) Then tableau(i, j) = -1 * tableau(i, j)
        End If
    Next i
End Sub
```

2.5.6 Korak 2: konstruiraj začetno tabelo

Začetno tabelo konstruiramo tako, da matriko koeficientov dopolnimo z vrstico koeficientov namenske funkcije in stolpcem desnih strani neenačb. V začetno množico baznih vektorjev damo samo dopolnilne spremenljivke.

Konstruiranje začetne tabele se izvede s klicem procedure *DoStep2* (Slika 16).

Slika 16: Implementacija potisni-povleci: konstruiranje začetne tabele

```
Private Sub DoStep2()

    j = S

    For i = 1 To M

        If Definition.ConstraintType(i) <> "=" Then
            j = j + 1

            If (Definition.ConstraintType(i) = "<") And (Definition.RHS(i) >= 0) Then
                BVS(i) = j
            End If

        End If
    Next i

    Result.Status = NotCalculated

End Sub
```

2.5.7 Korak 3: napolni množico baznih vektorjev

Korak 3 se izvaja s klicem procedure *DoStep3* po naslednjih korakih (Slika 17):

1. izbor nove vstopne spremenljivke *k*: naj bo tista pri največjem *Cj*,
2. kreiranje modificirane tabele (počisti vrstice, ki že imajo *BVS*),
3. izračun pivotne vrstice *r* s pomočjo *CR* testa,
4. obravnavanje rezultata *CR* testa: če ni rezultata, vzemi naslednji največji *Cj*,
5. če je *BVS(r)* prazen, izvedi pivotiranje in končaj izvajanje koraka 3, sicer
6. ponovi z naslednjim največjim *Cj*.

Slika 17: Implementacija potisni-povleci: polnjenje BVS (korak 3)

```
Private Sub DoStep3()

    Do While Not sortedCj.IsEmpty

        k = sortedCj.Largest.j                                                    (1)

        modTableau = ModuleUtilities.CloneMatrix(tableau)                         (2)
        For i = 1 To M
            If BVS(i) <> 0 Then
                For j = 1 To S
                    modTableau(i, j) = 0
                Next j
            End If
        Next i

        r = ModuleGauss.ColumnRatioTest(Inc, modTableau, k, degeneracyOccured, True)   (3)

        If r = 0 Then                                                             (4)
            sortedCj.RemoveLargest
        Else

            If BVS(r) = 0 Then                                                    (5)
                ModuleGauss.GaussPivot Inc, tableau, k, r, BVS
                Exit Do
            Else                                                                  (6)
                sortedCj.RemoveLargest
            End If

        End If
    Loop
End Sub
```

2.5.8 Korak 4: potisni dalje proti optimalni rešitvi

Korak 4 se izvaja s klicem procedure *DoStep4* po naslednjih korakih (Slika 18):

1. izbor naslednje vstopne spremenljivke: naj bo tista, pri kateri je *Cj* največji in pozitiven,
2. izračun pivotne vrstice *r* s pomočjo *CR* testa
3. obravnavanje rezultata *CR* testa: če ni rezultata, označi rešitev za neomejeno, sicer
4. izvedi pivotiranje.

Slika 18: Implementacija potisni-povleci: potisni dalje proti optimalni rešitvi (korak 4)

```
Private Sub DoStep4()

    k = sortedCj.Largest.j                                                    (1)

    r = ModuleGauss.ColumnRatioTest(Inc, tableau, k, degeneracyOccured)       (2)

    If r = 0 Then                                                             (3)
        Result.Status = Unbounded
        Exit Sub
    End If
    ModuleGauss.GaussPivot Inc, tableau, k, r, BVS                            (4)

End Sub
```

2.5.9 Korak 5: preveri trenutno rešitev

Korak 5 se izvaja s klicem funkcije *DoStep5()*, ki kot rezultat vrača obvestilo o rezultatu preverjanja rešitve.

Pri tem je preverjanje implementirano po naslednjih korakih (Slika 19):

1. preverjanje obstoja negativne vrednosti *RHS*,
2. preverjanje obstoja *Cj*, katerega vrednost je večja od nič,
3. v primeru, da sta oba pogoja pozitivna, je rešitev optimalna: zaključi izvajanje algoritma,
4. v primeru, da ne glede na prvega, drugi pogoj ni izpolnjen, obstaja boljša rešitev: nadaljuj s korakom 4 in
5. v primeru, da prvi pogoj ni izpolnjen, drugi pa je, je rešitev šla predaleč: nadaljuj s korakom 6.

Slika 19: Implementacija potisni-povleci: preverjanje trenutne rešitve (korak 5)

```
Private Function DoStep5() As String
    Dim decision As String: decision = ""
    sortedCj.RebuildWith tableau
    decision = decision & IIf(negativeRHS, "F", "T")                    (1)
    decision = decision & IIf(sortedCj.Positive, "F", "T")              (2)
    Select Case decision
          Case "TT": stepName = "OPTIMAL"                               (3)
          DoStep5 = "All RHS >= 0 and all Zj <= 0 : solution is optimal"
       Case "TF", "FF": stepName = "4-PUSH"                             (4)
          DoStep5 = "Not all Zj <= 0: continue with PUSH (Step 4)"
       Case "FT": stepName = "6-PULL"                                   (5)
          DoStep5 = "Not all RHS >= 0: continue with PULL (Step 6)"
    End Select
End Function
```

2.5.10 Korak 6: povleci rešitev nazaj na področje izračunljivosti

Korak 6 se izvaja s klicem procedure *DoStep6*, z naslednjimi koraki (

Slika *20*):

1. ugotavljanje novih *k* in *r* s pomočjo *RR* testa,
2. obravnavanje rezultata *RR* testa: če rezultata ni, je rešitev neizvedljiva, sicer
3. izvedba pivotiranja in nadaljevanje izvajanja algoritma.

Slika 20: Implementacija potisni-povleci: povleci na področje izračunljivosti (korak 6)

```
Private Sub DoStep6()
    ModuleGauss.RowRatioTest Inc, tableau, k, r                         (1)
    If r = 0 Then                                                       (2)
        Result.Status = Infeasible: Exit Sub
    End If
    ModuleGauss.GaussPivot Inc, tableau, k, r, BVS                      (3)

End Sub
```

2.5.11 Pomožne funkcije in procedure modula

Za lažje izvajanje algoritma so v modulu izvedene še naslednje pomožne funkcije in procedure:

1. *CompleteBVS*() – funkcija, ki preveri, ali ima *BVS* napolnjene vse vrednosti,

2. *NegativeRHS*() – funkcija, ki preveri, ali v *RHS* obstajajo negativne vrednosti.

3 PROGRAMSKO ORODJE SIXPAP

Namen orodja je računalniška implementacija algoritmov potisni-povleci in standardni simpleks ter primerjava učinkovitosti med obema ob reševanju posameznega ali cele vrste problemov linearnega programiranja. Orodje omogoča enostavno in uporabniku prijazno reševanje problemov z eno, drugo ali obema metodama hkrati.

Orodje se imenuje SixPap, saj prvi del imena "Six" nekako namiguje na simpleks, drugi del "Pap" pa na potisni-povleci (angl. *push–and–pull*) metodo. V celoti je kreirano in zgrajeno z razvojnim orodjem MicroSoft Visual Studio 6.0 s programskim jezikom MicroSoft Visual Basic.

Orodje ima v svojem naboru kar precej primerov problemov LP, katerih koren imena predstavlja avtorja oziroma vir. To so:

1. Bastic, 9 primerov (Bastič & Meško, 1979),
2. Cottle, 1 primer (Arsham, 1994-2015),
3. Enge_Huhn, 1 primer (Enge & Huhn, 1998),
4. Example, 16 primerov (Arsham, Baloh, Damij, & Grad, 2003),
5. Geary, 2 primera (Geary & Spencer, 1973),
6. Klee_Minty, 1 primer (Klee & Minty, 1972) ,
7. Tijessen, 1 primer (Arsham, 1994-2015) in
8. Vadnal, 4 primeri (Vadnal, 1977).

3.1 Koncept Sistema

Sistem ima uporabniku prijazen uporabniški vmesnik za kreiranje problemov linearnega programiranja. Probleme je možno v sistem dodati tudi z ustrezno t. i. definicijsko datoteko, ki je tipa tekst in zadošča enostavnim strukturnim pravilom. Več o podatkovni organizaciji in upravljanju podatkov je prikazano v naslednjih poglavjih.

Učinkovitost algoritmov se poleg uspešnosti pri iskanju rešitve in štetja iteracij, ki so potrebne, da pridemo do konca izvajanja, meri še s štetjem osnovnih računalniških operacij:

1. seštevanje in odštevanje,
2. množenje in deljenje,
3. število ponovitev v zankah in
4. število odločitvenih operacij.

Poleg tega sistem kot rezultat prikaže podroben in nazoren potek izvajanja algoritma z vsemi vmesnimi tabelami ter opisom ugotovitve algoritma v smislu izvajanja naslednjega koraka. Pri tem sta navedena tudi indeksa ključnega elementa, *k* in *r*, za izračun naslednje tabele.

Na koncu izvajanja sledi enostaven in nazoren preračun in testiranje rešitve glede na pogojne neenačbe, ki dokazujejo pravilnost rešitve. V povzetku (angl. *summary*) je poleg statusa rešitve problema, uspešnosti testiranja na pogojne neenačbe, numerične rešitve problema in vrednosti mer učinkovitosti še status, ki označuje degeneracijo.

Sistem je objektno orientiran, njegova funkcionalnost temelji na objektni strukturi, ki je sestavljena iz glavnih funkcionalnih objektov in uporabnih podpornih objektov. Sistem je tudi dogodkovno gnan: objekti komunicirajo z uporabniškim vmesnikom in med sabo, v glavnem s pomočjo proženja in zaznavanja sistemskih dogodkov. Posamezni objekti so podrobneje opredeljeni v naslednjih poglavjih.

Uporabnik s sistemom komunicira preko grafičnega uporabniškega vmesnika. To je večokenski vmesnik, sestavljen iz treh glavnih oken, od katerih ima vsako svojemu namenu ustrezno funkcionalnost in pripadajoče elemente.

Večinoma so elementi t. i. uporabniške kontrole (angl. *user control*), narejene po meri točno določenega problema oz. z namenom izvedbe točno določene funkcionalnosti. To pripomore k povečanju učinkovitosti in uporabniku bolj prijaznemu uporabniškemu vmesniku. Temu je dodano še učinkovito obvladovanje sistemskih in logičnih napak s pomočjo t. i. lovilcev napak (angl. *error handling routines*), kar naredi sistem bolj stabilen in robusten. Uporabniški vmesnik s svojimi elementi je podrobneje prikazan v nadaljevanju.

V sistem je integrirana tudi pomoč za uporabnika, ki je enostavno dostopna in uporabniku nudi potrebne informacije za uporabo in razumevanje sistema.

3.2 Organizacija in upravljanje podatkov

Podatki so organizirani v tri skupine: definicijski podatki, podatki o povzetku reševanja in podatki o podrobnem poteku reševanja problema. Shranjeni so v dveh datotekah, katerih organizacija je podrobneje razložena v nadaljevanju.

Definicija problema in povzetek zadnjega reševanja sta shranjena v navadni tekstovni datoteki. Prvi del datoteke zajema splošne podatke o obravnavanem problemu. Ti podatki so:

1. ime problema,

2. vir in
3. splošne opombe.

Drugi del zajema podatke o:

1. namenski funkciji s pripadajočimi koeficienti in o številu spremenljivk ter
2. pogojnih neenačbah s pripadajočimi koeficienti, relacijami neenačb, o vrednostih desnih strani in njihovem številu.

Tretji oziroma četrti del datoteke se kreira dinamično ob izvajanju enega, drugega ali obeh algoritmov. Ti podatki predstavljajo povzetek izvajanja algoritmov. Sestojijo iz:

1. imena uporabljenega algoritma,
2. statusa rešitve,
3. optimalne vrednosti namenske funkcije, če ta obstaja,
4. vrednosti elementov *BVS*,
5. vrednosti spremenljivk pri optimalni rešitvi,
6. indikatorja degeneracije,
7. vrednosti mer učinkovitosti:
   a) števila iteracij,
   b) števila seštevanj in odštevanj,
   c) števila množenj in deljenj,
   d) števila zank in
   e) števila odločitev,
8. datuma in časa izvedbe algoritma ter
9. imena datoteke, kjer so shranjeni podrobni podatki o poteku izvajanja algoritma.

Datoteka s podatki o podrobnem poteku izvajanja prikazuje dejansko izvajanje algoritma, korak za korakom. Pri tem se beležijo vsi relevantni podatki, kot so trenutno stanje *BVS*, tabela, izstopna in vstopna spremenljivka ter komentarji s podrobnim opisom prejšnje in naslednje akcije algoritma in razlago za odločitev za vsako iteracijo. Pri konkretnem problemu lahko obstajata dve taki datoteki, po ena za vsako metodo.

Podatkovne datoteke in delo z njimi uravnava in nadzoruje podsistem, imenovan odlagališče (angl. *Repository*). Uporabnik s tem podsistemom komunicira preko posebnega okna uporabniškega vmesnika, ki je podrobneje opisano v poglavju o uporabniškem vmesniku.

To pomeni, da se uporabniku ni treba ukvarjati z operacijami, ki so potrebne za delo z datotekami, podatkovno strukturo in lokacijami datotek v datotečnem sistemu. Za vse to poskrbi podsistem odlagališče, vključno s kreiranjem potrebnih manjkajočih podmap datotečnega sistema, ki je del operacijskega sistema računalnika.

3.3 Implementacija algoritmov

Implementacija algoritmov je izpeljana v dveh glavnih in enem pomožnem modulu:

1. modul **Potisni-Povleci**, v katerem je implementiran iterativni postopek izvajanja algoritma potisni-povleci z vsemi pripadajočimi koraki in kontrolami,
2. modul **Simpleks**, v katerem je implementiran iterativni postopek izvajanja algoritma standardne simpleks metode z vsemi pripadajočimi koraki in kontrolami ter
3. pomožni modul **Gauss**, v katerem so implementirani postopki za izračun ustrezne nove tabele, ki temelji na operacijah za pivotiranje vrstic po Gaussu.

Uporaba in izvajanje modulov je podrobneje opisana v poglavjih o računalniški implementaciji algoritmov v okviru razlage posamezne metode.

Sistem do modula dostopa z implementacijo izbranega algoritma preko klica ene same procedure, ki služi kot vstopna točka modula. Pri tem se prenese en sam parameter, ki predstavlja kazalec na krovni objekt problema. Ta nosi vso potrebno podatkovno strukturo problema, beleži potek in rezultat izvajanja algoritma in komunicira z ostalimi objekti in uporabniškim vmesnikom. Ob koncu izvajanja algoritma se rezultat preko tega objekta prenese nazaj v sistem.

Prednost modularnega pristopa je v možnosti uporabe tehnologije dinamičnih knjižnic (angl. *Dynamic Link Library*, v nadaljevanju DLL). Z nekaj manjšimi popravki pri naslednji verziji sistema lahko uporabnik ob upoštevanju predpisane objektne strukture sam pripravi svojo implementacijo algoritma, jo prevede v DLL datoteko in poveže v sistem. S tem je odprta možnost nadgradnje sistema s praktično neomejenimi variantami implementacij algoritmov brez posega v sam sistem.

Programska koda implementacije modulov se nahaja v naslednjih datotekah:

1. modul Potisni-Povleci, v katerem je implementiran iterativni postopek izvajanja algoritma potisni-povleci z vsemi pripadajočimi koraki in kontrolami, katerega implementacija se nahaja v datoteki ModulePushPull.bas z izpisom kode v Prilogi 1,
2. modul Simpleks, katerega implementacija se nahaja v datoteki ModuleSimplex.bas z izpisom kode v Prilogi 2 ter
3. pomožni modul Gauss, katerega implementacija se nahaja v datoteki ModuleGauss.bas z izpisom kode v Prilogi 3.

3.4 Objektna struktura

Objektna struktura je sestavljena iz treh glavnih, med seboj povezanih objektov in nekaj uporabnih podpornih objektov, ki olajšajo delovanje posameznih kompleksnejših funkcij.

3.4.1 Glavni objekti

Namen glavnih objektov je obvladovanje in upravljanje dela s problemom linearnega programiranja kot s celoto, od definicije problema do rezultatov, ki jih data tako potisni-povleci kot standardni simpleks algoritem. Ti objekti so:

1. objekt Problem,
2. objekt Definicija Problema ter
3. objekt Rezultat Problema.

3.4.1.1 Objekt Problem

Problem je implementiran kot objekt razreda *ClassProblem* in je nadrejen objektu Definicija Problema, ki je instanca razreda *ClassProblemDefinition,* ter dvema objektoma Rezultat Problema razreda *ClassProblemResult,* pri čemer predstavlja eden od obeh objektov rezultat problema izračunan po metodi simpleks, drugi pa rezultat problema, izračunan po metodi potisni-povleci.

Razred *ClassProblem* predstavlja strukturo in funkcionalnost krovnega objekta problema in kot tak vsebuje lastnosti in metode, ki se tičejo obvladovanja problemov na najvišjem nivoju. To so lastnosti in metode, ki nastopajo pri operacijah, kot so kreiranje objektne strukture novega problema in njena inicializacija, metode za branje in zapisovanje podatkov v in iz datotek, torej sodelovanje z datotečnim sistemom, ter metode, ki upravljajo in nadzirajo delovanje različnih prikazov podatkov obravnavanega problema.

Razred vsebuje naslednje lastnosti:

1. *Definition* – objekt razreda *ClassProblemDefinition*, ki nosi definicijske podatke problema. Podrobneje je opisan v nadaljevanju;
2. *Result*(*method*) – niz z dvema elementoma, od katerih pripada eden metodi simpleks, drugi pa metodi *potisni-povleci*. Elementa sta objekta razreda *ClassProblemDefinition*, ki je podrobneje opisan v nadaljevnaju;
3. *ProblemSaved* – status shranjevanja problema;
4. *IsInAnalysis* – indikator udeležbe problema v skupinski analizi;
5. *AnalysisRowIndex* – indeks problema v tabeli pri analizi več problemov na enkrat.

Osnovna funkcionalnost razreda je poleg metod podrejenih razredov opredeljena z naslednjimi metodami:

1. *CreateNew* (*newProblemName*) – kreiranje novega problema;
2. *OpenProblem* (*problemFileName*) – odpiranje problema na podlagi datoteke;
3. *SaveProblem* (*problemFileName*) – shranjevanje problema v datoteko;

4. *CalcDifference* () – izračun razlike števcev učinkovitosti metode potisni-povleci in standardne simpleks metode;
5. *DisplayInAnalysis* () – prikaz problema z rešitvami v tabeli skupinske analize.

Programska koda implementacije razreda se nahaja v datoteki ClassProblem.cls. Primer izpisa definicijske datoteke s povzetkom rezultatov je v Prilogi 4.

3.4.1.2 Objekt Definicija Problema

Objekt Definicija Problema predstavlja instanco razreda *ClassProblemDefinition*, ki je podrejen razredu *ClassProblem*. Njegova funkcija je upravljanje in nadzor dela z vsemi potrebnimi podatki, ki se tičejo definicije problema. Podpira tako operacije kot sta branje in pisanje v definicijsko datoteko (v sodelovanju z nadrejenim razredom) kot tudi operacije, povezane z ažuriranjem definicijskih podatkov z vsemi potrebnimi validacijskimi mehanizmi.

Informacije, ki jih objekt tega razreda vsebuje, se torej nanašajo na definicijo obravnavanega problema linearnega programiranja. Razred ima naslednje lastnosti:

1. *DefinitionFile* – polno ime datoteke, ki vsebije definicijo problema,
2. *ProblemName* – naziv problema, ki je istočasno ime datoteke,
3. *Source* – izvor definicije problema,
4. *Comment* – priložnostni komentar k problemu,
5. *M* – število pogojnih neenačb,
6. *S* – število dopolnilnih spremenljivk,
7. *Objective* – vrsta namenske funkcije, iskanje minimuma ali maksimuma (*MIN*, *MAX)*,
8. *ConstraintType*() – niz tipov pogojnih neenačb (<, =, >),
9. *Cj*() – niz vrednosti Cj spremenljivk namenske funkcije,
10. *RHS*() – niz vrednosti desnih strani neenačb (angl. *right hand side*, kratica RHS) in
11. *A*() – matrika koeficientov neenačb problema.

Osnovna funkcionalnost razreda je opredeljena z naslednjimi metodami:

1. *ResetToDefault*(*newProblemName*) – kreiranje nove prazne definicije problema LP,
2. *ReadFromFile*(*problemFileName*) – branje podatkov definicije problema iz datoteke in
3. *SaveDefinition*(*problemFileName*) – shranjevanje definicije problema v datoteko.

Programska koda implementacije razreda se nahaja v datoteki ClassProblemDefinition.cls.

3.4.1.3 Objekt Rezultat Problema

Objekt Rezultat Problema predstavlja instanco razreda *ClassProblemResult*, ki je podrejen razredu *ClassProblem*. Pri konkretnem problemu ima lahko dva pojavka (objekta), od katerih eden pripada potisni-povleci, drugi pa standardni simpleks metodi.

Upravlja z detajlnimi podatki, ki se nanašajo na reševanje problema. Operacije, ki jih podpira, zajemajo tako branje in pisanje v pripadajoče datoteke kot tudi komunikacijo z ustreznimi okenskimi kontrolami, ki skrbijo za prikaz rezultatov.

Vsebovane informacije se torej nanašajo na izvajanje enega in drugega algoritma, tako v podrobnostih poteka kot tudi v povzetku rezultata. Razred ima naslednje lastnosti:

1. *MethodName* – ime uporabljene metode,
2. *Counters* – objekt pomožnega razreda *ClassCounters*, ki vsebuje vse potrebne števce in metode za ugotavljanje učinkovitosti izvajanja algoritma,
3. *StartedWhen* – časovna značka začetka izvajanja algoritma,
4. *BVS*() – niz baznih vektorjev (angl. *Base Vector Set*, kratica BVS),
5. *ConstraintTest*() – niz rezultatov testiranja rešitve po posameznih omejitvenih funkcijah,
6. *Xj*() – niz vrednosti spremenljivk optimalne rešitve,
7. *DetailFile* – polno ime datoteke, v kateri je zabeležen podroben potek izvajanja algoritma,
8. *Status* – status izvajanja algoritma,
9. *FunctionValue* – vrednost namenske funkcije,
10. *TestResult* – rezultat testiranja ustreznosti rešitve glede na namensko funkcijo in pogoje,
11. *Degeneracy* – indikator ugotovljene degeneracije,
12. *OutputDetailFile* – objekt razreda ClassOutputDetailFile, ki skrbi za delo z datoteko, v kateri je zapisan podroben potek izvajanja. Razred je podrobneje opisan v poglavju o pomožnih objektih.

Osnovna funkcionalnost razreda je opredeljena z naslednjimi metodami:

1. *ShowResult* () – prikaz rezultata izvajanja algoritma,
2. *ReadResultData* () – branje podatkov o rešitvi in izvajanju algoritma iz datotek,
3. *SaveResultData* () – shranjevanje rešitve in poteka izvajanja v datoteki.

Programska koda implementacije razreda se nahaja v datoteki ClassProblemResult.cls.

3.4.2 Pomožni objekti

Funkcionalnost sistema je povečana z uporabo različnih uporabnih pomožnih objektov kot so različni kompleksni števci, urejeni seznami in objekti za delo z datotekami s podrobnim potekom izvajanja algoritmov. Ti objekti so:

1. objekt Števci,
2. objekt Datoteka Podrobnosti,
3. objekta Urejeni Seznam in Element Urejenega Seznama.

3.4.2.1 Objekt Števci

Objekt Števci je predstavnik razreda *ClassCounters* in sodeluje z objektom Rezultat Problema. Njegova vloga je, da hrani podatke o in upravlja z različnimi števci, ki merijo učinkovitost algoritma. To so:

1. števci iteracij,
2. števci seštevanj in odštevanj,
3. števci množenj in deljenj,
4. števci prehodov zank in
5. števci izvajanj odločitev.

Programska koda implementacije razreda se nahaja v datoteki ClassCounters.cls.

3.4.2.2 Objekt Datoteka Podrobnosti

Tudi objekt Datoteka Podrobnosti, ki je opredeljen z razredom *ClassOutputDetailFile*, sodeluje z objektom Rezultat Problema. Njegova naloga je obvladovanje operacij, potrebnih za sprotno zapisovanje poteka izvajanja algoritmov v za to namenjeno datoteko. Rezultat delovanja je torej datoteka s podrobnim potekom izvajanja posameznega algoritma za izbrani problem linearnega programiranja. Programska koda implementacije razreda se nahaja v datoteki ClassOutputDetailFile.cls. Primera izpisa podrobnega poteka algoritma potisni-povleci in simpleks sta v Prilogi 5 in Prilogi 6.

3.4.2.3 Objekta Urejeni Seznam in Element Urejenega Seznama

Objekta Urejeni Seznam in Element Urejenega Seznama sodelujeta samo pri izvajanju algoritma potisni-povleci. Pri tem predstavlja objekt razreda *ClassList* urejeni seznam, množica objektov razreda *ClassItem* pa elemente tega seznama.

Njun namen je vzdrževati urejen, po velikosti padajoč seznam vrednosti C_j, ki služi lažjemu ugotavljanju naslednje vstopajoče spremenljivke pri koraku 3 – napolni *BVS*.

Njuna medsebojna relacija je je-del (angl. *is–part–of*), kot celota pa sicer nista povezana z nobenim drugim objektom. Programska koda implementacije obeh razredov se nahaja v datotekah ClassList.cls in ClassItem.cls.

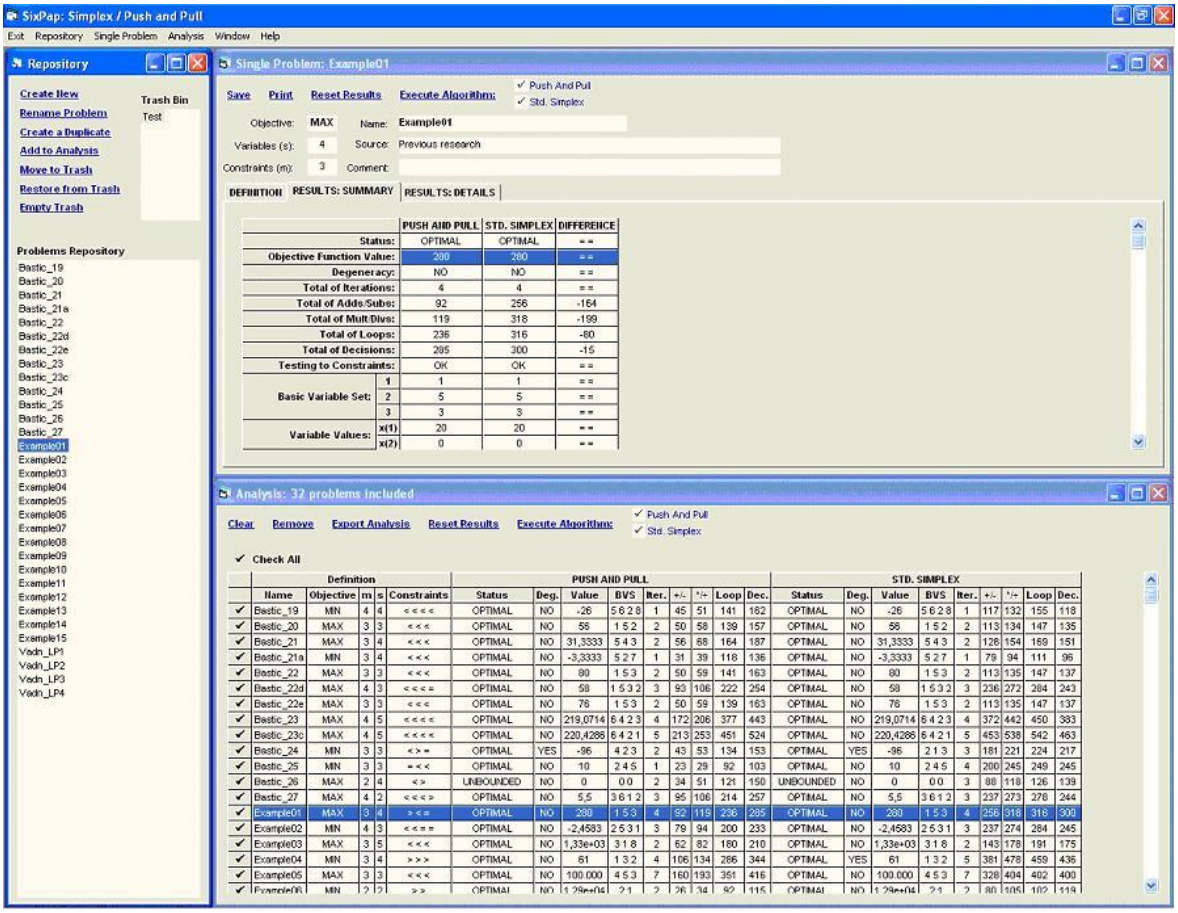
3.5 Uporabniški vmesnik

Uporabniški vmesnik sistema je tako imenovani večokenski uporabniški vmesnik. Kot glavno okno je uporabljen Microsoft Visual Studio grafični objekt vrste MDI Form (Multi Document Interface). Ta služi kot lupina za tri glavna okna (Slika 21):

1. Odlagališče,
2. Posamezni Problem (angl. *single problem*) in
3. Analiza (angl. *analysis*).

Poleg glavnih oken nosi tudi vrstico z dinamičnim menijem. Vse funkcije, opisane v nadaljevanju, so podrobno razložene tudi v datoteki z uporabniško pomočjo, ki je sestavni del sistema.

Slika 21: Ekranska slika uporabniškega vmesnika orodja SixPap



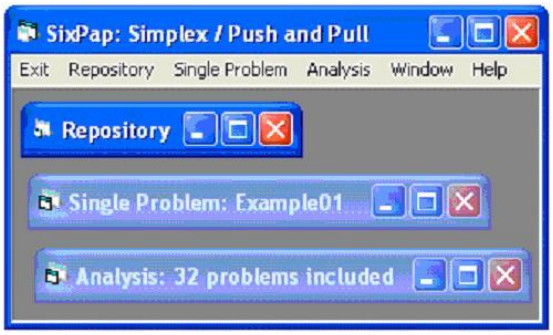
3.5.1 Glavno okno

Glavno okno – lupina sistema SixPap – je, kot že rečeno, sestavljeno iz treh glavnih elementov, okna z Odlagališčem, okna Posameznega Problema in okna z Analizo, ki so podrobneje opisana v nadaljevanju. Poleg tega nosi tudi dinamični meni. Opcije dinamičnega menija predstavljajo po vrsti:

1. Izhod (angl. *exit*) – izhod iz aplikacije,
2. Odlagališče (angl. *repository*) – aktiviranje okna Odlagališča,
3. Posamezni Problem (angl. *single problem*) – aktiviranje okna Posameznega Problema,
4. Analiza (angl. *analysis*) – aktiviranje okna Analize,
5. Okno (angl. *window*) – vsebuje različne opcije za razporejanje glavnih treh oken znotraj lupine,
6. Pomoč (angl. *help*) – dostop do uporabniške pomoči.

Slika 22 prikazuje glavno okno z njegovimi elementi.

Slika 22: Ekranska slika glavnega okna z osnovnimi elementi



3.5.2 Okno odlagališča

Okno odlagališča služi kot vmesnik med uporabnikom in podsistemom Odlagališče, ki, kot že omenjeno, služi delu z vsemi potrebnimi datotekami (Slika 23).

Podsistem deluje na nivoju problema (kar pomeni, da operacije, ki jih izvaja, delujejo na izbranem problemu kot na celoti) in se ne spušča v podrobnosti.

Uporabnik ukaze za izvajanje operacij podsistemu pošilja preko treh glavnih elementov okna:

1. menija,
2. seznama, imenovanega Koš (angl. *trash bin*) in
3. seznama, imenovanega Odlagališče problemov.

Slika 23: Ekranska slika okna odlagališča

3.5.2.1 Meni odlagališča

Meni, preko katerega je mogoče izvajati operacije podsistema Odlagališče, vsebuje naslednje opcije:

1. Novo (angl. *create new*) – aktivira operacije, potrebne za kreiranje novega problema. Po opravljenem delu preda kontrolo oknu Posamezni problem, kjer se izvede podrobna definicija danega novega problema.
2. Preimenuj (angl. *rename problem*) – aktivira operacije, potrebne za spremembo imena problema. Nanaša se na trenutno izbrani (prvi) problem v seznamu Odlagališče problemov. Podsistem Odlagališče poskrbi za vsa potrebna preimenovanja podatkovnih datotek in ažuriranje podatkov v pripadajočih objektih.
3. Dupliciraj (angl. *create a duplicate*) – opcija, ki je koristna, kadar je novi problem precej podoben kateremu izmed obstoječih, saj pod novim imenom kreira natančno kopijo problema, ki je trenutno (prvi) izbran v seznamu Odlagališče problemov. Podsistem poskrbi za vse potrebno, da je zahteva realizirana.
4. Dodaj k analizi (angl. *add to analysis*) – aktivira vse potrebne operacije podsistema za dodajanje označenih problemov v seznam izbranih problemov okna.
5. Prestavi v koš (angl. *move to trash*) – izvede vse potrebne operacije za prenos izbranih problemov v seznam Koš (angl. *trash bin*). Te operacije dejansko prenesejo vse izbranim problemom pripadajoče datoteke v posebno datotečno mapo.
6. Vrni iz koša (angl. *restore from trash*) – je obraten postopek zgoraj omenjenemu. Tako se iz seznama Koš izbrani problemi prenesejo nazaj v seznam Odlagališče problemov, vključno s fizičnim premikanjem izbranih datotek.
7. Izprazni koš (angl. *empty trash*) – povzroči, da se seznam Koš izprazni, kar pomeni tudi fizično brisanje vsebine ustrezne datotečne mape.

3.5.2.2 Seznam Koš

Seznam služi kot koš za smeti. Vanj se prestavijo problemi, predvideni za brisanje. Podsistem Odlagališče omogoča vračanje izbranih problemov nazaj v aktivni seznam, pa tudi praznjenje koša, kar pomeni fizično brisanje vseh datotek, ki pripadajo izbranim problemom.

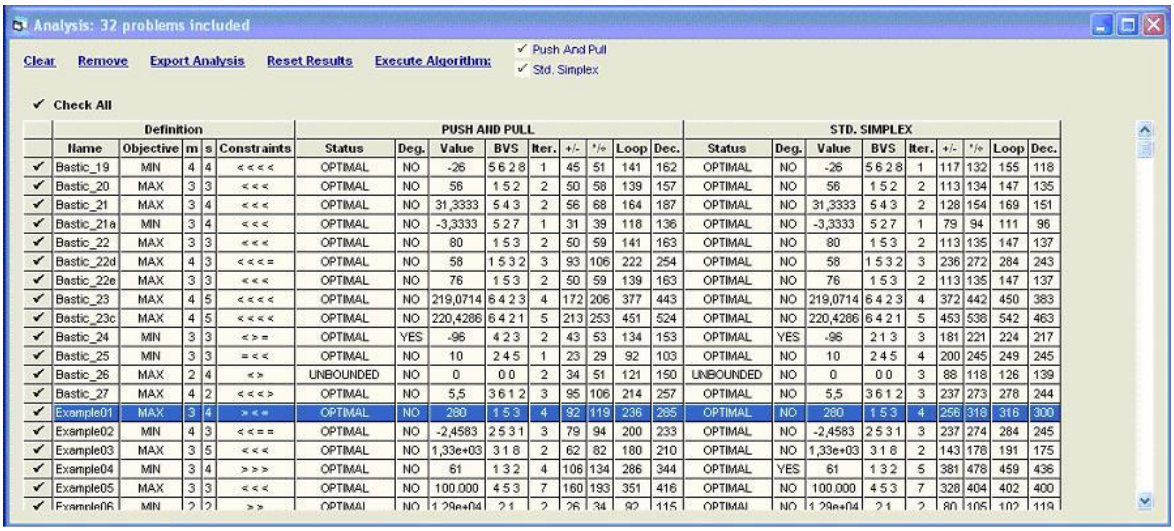
3.5.2.3 Seznam Odlagališče problemov

Predstavlja seznam problemov, ki so tako ali drugače zabeleženi v sistemu. Z izbiro elementa v tem seznamu podsistem Odlagališče poskrbi za prikaz vseh podrobnih podatkov (prvega, saj je možno izbrati več elementov) izbranega problema v oknu Posamezni problem. Oba seznama omogočata izbiranje elementov po standardih MicroSoft okolja in temeljita na posebej za to narejeni uporabniški grafični kontroli (angl. *user control*), imenovani MyFileListView. Ker omenjena kontrola poleg osnovnih operacij izbiranja v seznamu in osnovnih operacij pregledovanja vsebin datotečnih map podpira še operacije, ki so specifične za delo z datotečno organizacijo sistema SixPap, ter komunikacijo z ostalima dvema oknoma preko sistemove objektne strukture, postane razvoj aplikacije zelo pregleden in poenostavljen. Meni okna Odlagališče temelji na posebej za to narejeni uporabniški grafični kontroli, imenovani MyActiveLabels, ki poenostavi kreiranje in pozicioniranje menija ter vsebuje vse ključne akcije z miško. Tudi uporaba te kontrole v veliki meri poenostavi razvoj aplikacije.

3.5.3 Okno skupinske analize

Okno je namenjeno delu z večjim številom problemov.

Slika 24: Ekranska slika okna skupinske analize



	Definition					PUSH AND PULL								STD. SIMPLEX							
	Name	Objective	m	s	Constraints	Status	Deg.	Value	BVS	Iter.	+/-	*/=	Loop	Dec.	Status	Deg.	Value	BVS	Iter.	+/-	*/=	Loop	Dec.
✓	Bastic_19	MIN	4	4	< < < <	OPTIMAL	NO	-26	5 6 2 8	1	45	51	141	162	OPTIMAL	NO	-26	5 6 2 8	1	117	132	155	118
✓	Bastic_20	MAX	3	3	< < <	OPTIMAL	NO	56	1 5 2	2	50	58	139	157	OPTIMAL	NO	56	1 5 2	2	113	134	147	135
✓	Bastic_21	MAX	3	4	< < <	OPTIMAL	NO	31,3333	5 4 3	2	56	68	164	187	OPTIMAL	NO	31,3333	5 4 3	2	128	154	169	151
✓	Bastic_21a	MIN	3	4	< < <	OPTIMAL	NO	-3,3333	5 2 7	1	31	39	118	136	OPTIMAL	NO	-3,3333	5 2 7	1	79	94	111	96
✓	Bastic_22	MAX	3	3	< < <	OPTIMAL	NO	80	1 5 3	2	50	59	141	163	OPTIMAL	NO	80	1 5 3	2	113	135	147	137
✓	Bastic_22d	MAX	4	3	< < < =	OPTIMAL	NO	58	1 5 3 2	3	93	106	222	254	OPTIMAL	NO	58	1 5 3 2	3	236	272	284	243
✓	Bastic_22e	MAX	3	3	< < <	OPTIMAL	NO	76	1 5 3	2	50	59	139	163	OPTIMAL	NO	76	1 5 3	2	113	135	147	137
✓	Bastic_23	MAX	4	5	< < < <	OPTIMAL	NO	219,0714	6 4 2 3	4	172	206	377	443	OPTIMAL	NO	219,0714	6 4 2 3	4	372	442	450	383
✓	Bastic_23c	MAX	4	5	< < < <	OPTIMAL	NO	220,4286	6 4 2 1	5	213	253	451	524	OPTIMAL	NO	220,4286	6 4 2 1	5	453	536	542	463
✓	Bastic_24	MIN	3	3	< > =	OPTIMAL	YES	-96	4 2 3	2	43	53	134	153	OPTIMAL	YES	-96	2 1 3	3	181	221	224	217
✓	Bastic_25	MIN	3	3	= < <	OPTIMAL	NO	10	2 4 5	1	23	29	92	103	OPTIMAL	NO	10	2 4 5	4	200	245	249	245
✓	Bastic_26	MAX	2	4	< >	UNBOUNDED	NO	0	0 0	2	34	51	121	150	UNBOUNDED	NO	0	0 0	3	88	118	126	139
✓	Bastic_27	MAX	4	2	< < < >	OPTIMAL	NO	5,5	3 6 1 2	3	95	106	214	257	OPTIMAL	NO	5,5	3 6 1 2	3	237	273	278	244
✓	Example01	MAX	3	4	> < =	OPTIMAL	NO	280	1 5 3	4	92	119	236	285	OPTIMAL	NO	280	1 5 3	4	256	318	316	300
✓	Example02	MIN	4	3	< < = =	OPTIMAL	NO	-2,4583	2 5 3 1	3	79	94	200	233	OPTIMAL	NO	-2,4583	2 5 3 1	3	237	274	284	245
✓	Example03	MAX	3	5	< < <	OPTIMAL	NO	1,33e+03	3 1 8	2	62	82	180	210	OPTIMAL	NO	1,33e+03	3 1 8	2	143	178	191	175
✓	Example04	MIN	3	4	> > >	OPTIMAL	NO	61	1 3 2	4	106	134	286	344	OPTIMAL	YES	61	1 3 2	5	381	478	459	436
✓	Example05	MAX	3	3	< < <	OPTIMAL	NO	100.000	4 5 3	7	160	193	351	416	OPTIMAL	NO	100.000	4 5 3	7	328	404	402	400
✓	Example06	MIN	2	2	> >	OPTIMAL	NO	1,29e+04	2 1	2	26	34	92	115	OPTIMAL	NO	1,29e+04	2 1	2	80	105	102	119

Kot kaže Slika 24, okno poleg standardne naslovne vrstice, kjer je prikazano število trenutno sodelujočih problemov, vsebuje še dva glavna elementa:

1. meni in
2. seznam izbranih problemov.

3.5.3.1 Meni skupinske analize

Meni temelji na uporabniških kontrolah MyActiveLabels in MyCheckLabels. Prva je opisana že pri oknu odlagališča, namen druge pa je podoben, le da namesto izvajanja aktivnosti omogoča označevanje opcij. Opcije menija so:

1. Počisti (angl. *clear*) – izprazni seznam problemov sodelujočih v analizi,
2. Odstrani (angl. *remove*) – odstrani izbrani problem s seznama sodelujočih problemov,
3. Izvoz (angl. *export analysis*) – izvede operacije, potrebne za izvoz trenutnega seznama sodelujočih problemov z vsemi elementi v s tabulatorji ločeno tekstovno datoteko. Ta je primerna za nadaljnjo analizo s programskim orodjem za delo s tabelami, kot je na primer Microsoft Excel.
4. Resetiraj (angl. *reset results*) – izvede operacije, potrebne za resetiranje rezultatov s kljukico označenih problemov v seznamu. To pomeni, da se vsi stari rezultati izvajanja enega, drugega ali obeh algoritmov izbrišejo, ostanejo samo njihove definicije. Od označenih opcij v izbirnem meniju je odvisno, rezultati katere metode bodo izbrisani.
5. Izvrši (angl. *execute algorithm*) – aktivira izvajanje enega, drugega ali obeh algoritmov na v seznamu izbranih problemih. Kateri algoritem bo uporabljen, je podobno kot pri resetiranju rezultatov odvisno od označenih opcij v izbirnem meniju.
6. Potisni-povleci / simpleks (angl. *push and pull / std. simplex*) – izbirni meni, ki omogoča izbor algoritma, za katerega se bodo izvajale izbrane operacije.

3.5.3.2 Seznam izbranih problemov

Seznam temelji na uporabniški kontroli MyGrid. To je tabelarična kontrola, posebej prirejena za enostaven prikaz izbranih podatkov v obliki razpredelnice. Podpira vse potrebne operacije za izbiranje in označevanje zapisov.

Namen seznama je tabelarični prikaz problemov, ki nastopajo v večproblemski analizi. Za vsak prikazan problem vsebuje tri skupine podatkov: glavne definicijske podatke in značilne podatke rezultatov izvajanj obeh algoritmov. Tak način pomaga uporabniku na prvi pogled ugotavljati različne značilnosti obnašanja algoritmov pri posameznih problemih.

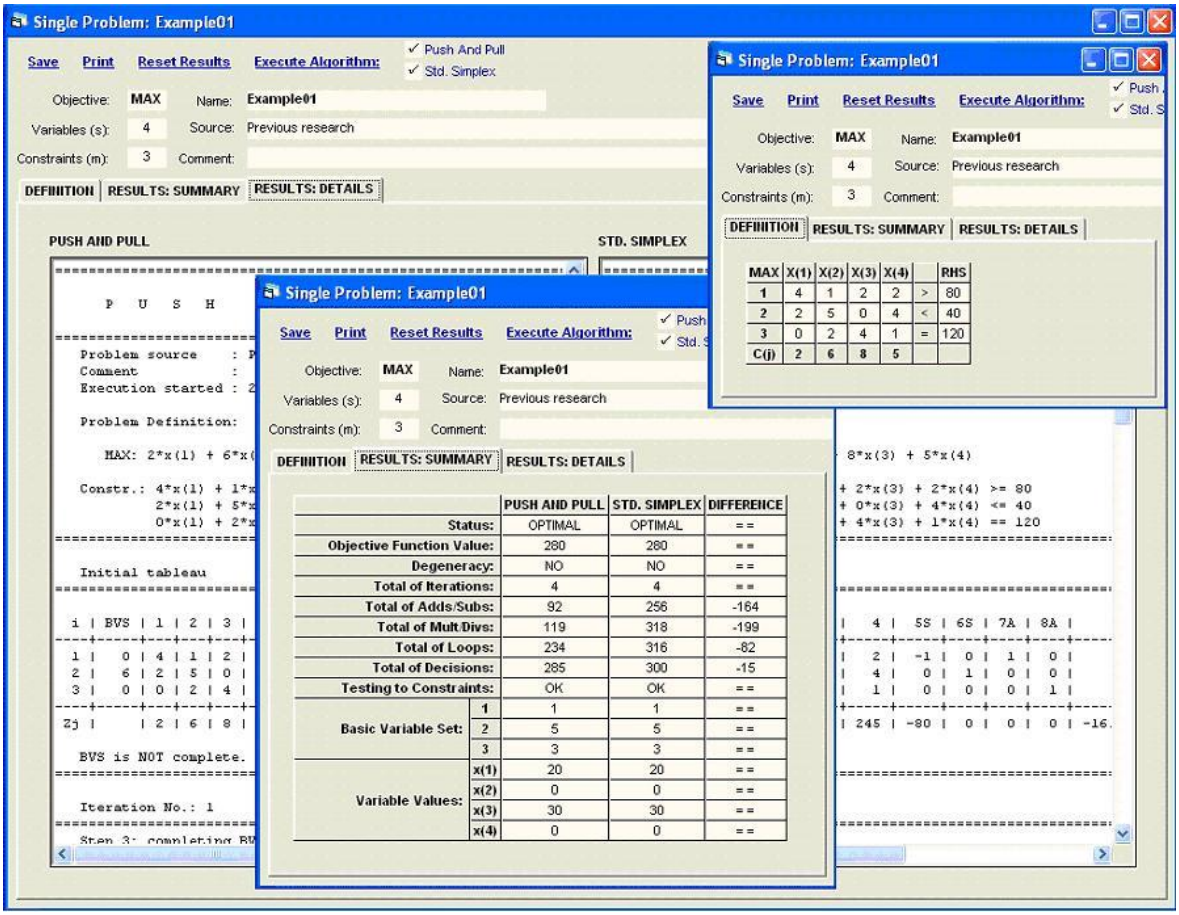
Seznam podpira dve vrsti izbiranja. Prva je označevanje s kljukico in pomeni izbor tistih problemov, nad katerimi se vršijo določene aktivnosti (resetiranje rezultatov, izvajanje

algoritmov). Druga je izbor posameznega problema, ki se posledično prikaže v oknu posameznega problema in je pripravljen za podrobnejši pregled in urejanje.

3.5.4 Okno posameznega problema

Okno posameznega problema je najbolj kompleksno okno sistema. Namenjeno je podrobnejšemu pregledu in delu s posameznim izbranim problemom.

Slika 25: Ekranska slika okna posameznega problema



S Slike 25 je razvidno, da okno posameznega problema poleg standardne naslovne vrstice, v kateri je prikazano ime izbranega problema, vsebuje še štiri glavne elemente:

1. meni,
2. definicijsko področje, ki je sestavljeno iz dveh delov,
3. področje s povzetkom razultatov in
4. področje s podrobnim potekom izvajanja algoritmov.

3.5.4.1 Meni posameznega problema

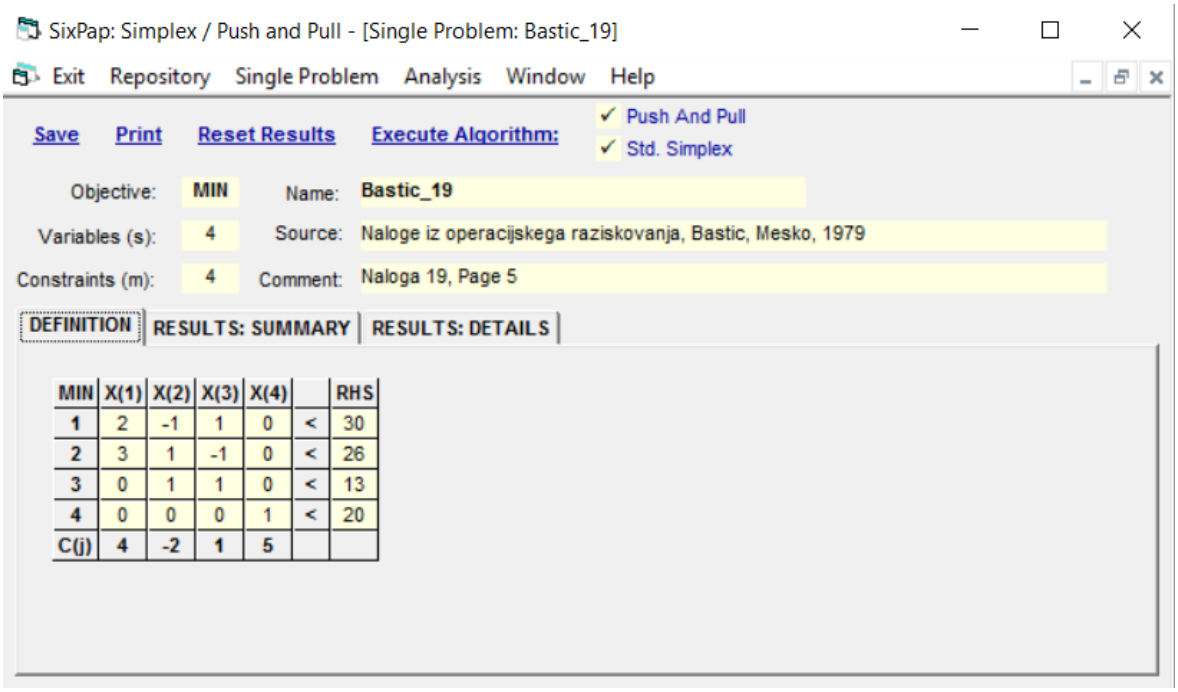
Tako kot meniji ostalih oken, tudi ta temelji na uporabniških kontrolah MyActiveLabels in MyCheckLabels. Opcije menija so:

- Shrani (angl. *save*) – shrani trenutno definicijo in opisne podatke problema,
- Natisni (angl. *print*) – aktivira pogovorno okno za tiskanje. To omogoča tiskanje definicije problema s povzetki rezultatov in/ali tiskanje podrobnega poteka izvajanja enega, drugega ali obeh algoritmov (glej priloge 4, 5 in 6),
- Resetiraj (angl. *reset results*) – izvede operacije, potrebne za resetiranje rezultatov izbranega problema. Rezultati katere metode bodo izbrisani, je odvisno od označenih opcij v izbirnem meniju,
- Izvrši (angl. *execute algorithm*) – aktivira izvajanje enega, drugega ali obeh algoritmov na izbranem problemu. Kateri algoritem bo uporabljen, je podobno kot pri resetiranju rezultatov odvisno od označenih opcij v izbirnem meniju,
- Potisni-povleci / simpleks (angl. *push and pull / std. simplex*) – izbirni meni, ki omogoča izbor algoritma, za katerega se bodo izvajale izbrane operacije.

3.5.4.2 Področje definicije problema

Področje je sestavljeno iz dveh delov: splošnih in podrobnejših podatkov (Slika 26).

Slika 26: Ekranska slika okna posameznega problema – definicijsko področje



MIN	X(1)	X(2)	X(3)	X(4)		RHS
1	2	-1	1	0	<	30
2	3	1	-1	0	<	26
3	0	1	1	0	<	13
4	0	0	0	1	<	20
C(j)	4	-2	1	5		

Prvi del zajema osnovne definicijske podatke (na Sliki 26 nad zavihki):

1. vrsta iskanega ekstrema (angl. *objective*) – maksimum ali minimum,
2. število spremenljivk namenske funkcije (*Slika 26*angl. *variables* (s)),
3. število pogojnih neenačb (*Slika 26*angl. *constraints* (m)); gre za število pogojnih neenačb brez splošnega pogoja $X_j \geq 0$),
4. polje z imenom problema (angl. *name*). Zaradi konsistentnosti je urejanje možno samo preko odlagališča) in
5. polji z navedbo vira ter dodatnim komentarjem k problemu (*Slika 26*angl. *source* in *comment*).

Drugi del definicijskega področja problema se nahaja na zavihku definicija (Slika 26, angl. *Definition*) in je prikazan kot razpredelnica. Za ta prikaz je uporabljena že opisana kontrola MyGrid z dodatno vkljopljeno možnostjo urejanja.

Dimenzija razpredelnice je določena avtomatično s spreminjanjem vrednosti števila spremenljivk namenske funkcije in števila pogojnih neenačb. Vrednosti znotraj razpredelnice predstavljajo vrednosti posameznih koeficientov pogojnih neenačb, njihove relacije (predzadnji stolpec) in vrednosti na desnih straneh (zadnji stolpec). Zadnja vrstica predstavlja vrednosti koeficientov namenske funkcije.

3.5.4.3 Področje s povzetkom rezultatov

Področje se nahaja na zavihku rezultati: povzetek (Slika 27, angl. *Results: Summary*) in je prikazano v razpredelnici, podprti s kontrolo MyGrid (tokrat brez vklopljene možnosti urejanja).

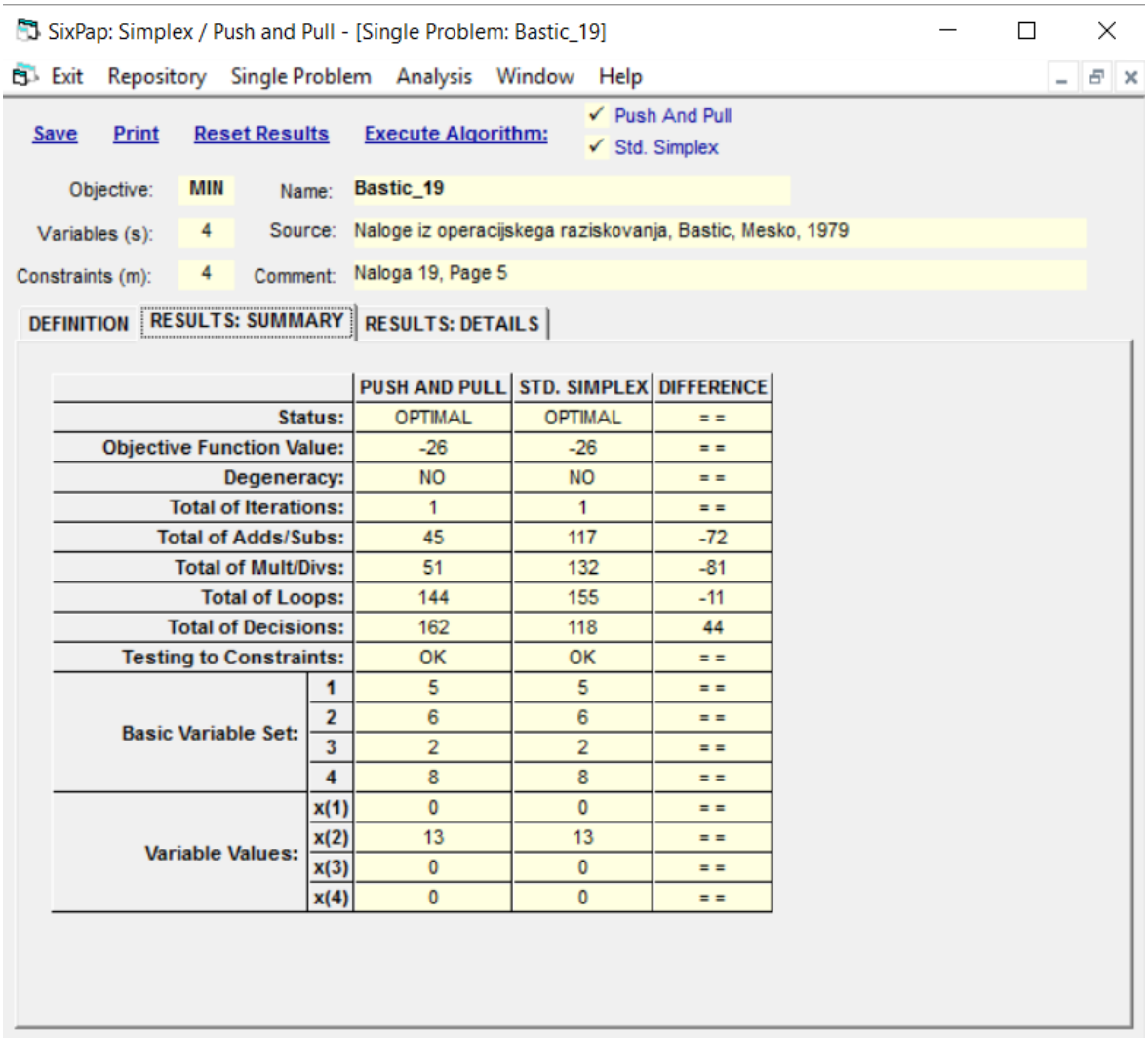
Prva dva stolpca razpredelnice predstavljata rezultate obeh algoritmov, zadnji pa razliko med njima. Tako je mogoče hitro ugotoviti učinkovitost enega in drugega algoritma.

V vrsticah razpredelnice so predstavljene spremenljivke, ki predstavljajo rezultat posameznega algoritma:

1. status (ali je algoritem prišel do optimalne rešitve ali ne),
2. optimalna vrednost namenske funkcije,
3. indikator degeneracije,
4. število iteracij,
5. število seštevanj in odštevanj,
6. število množenj in deljenj,
7. število ponavljanj zank,
8. število odločitev,
9. računalniški preračun ujemanja rezultata s pogojnimi neenačbami,

10. vrednosti množice baznih vektorjev in
11. optimalne vrednosti spremenljivk namenske funkcije.

Slika 27: Ekranska slika okna posameznega problema – povzetek rezultatov



		PUSH AND PULL	STD. SIMPLEX	DIFFERENCE
Status:		OPTIMAL	OPTIMAL	= =
Objective Function Value:		-26	-26	= =
Degeneracy:		NO	NO	= =
Total of Iterations:		1	1	= =
Total of Adds/Subs:		45	117	-72
Total of Mult/Divs:		51	132	-81
Total of Loops:		144	155	-11
Total of Decisions:		162	118	44
Testing to Constraints:		OK	OK	= =
Basic Variable Set:	1	5	5	= =
	2	6	6	= =
	3	2	2	= =
	4	8	8	= =
Variable Values:	x(1)	0	0	= =
	x(2)	13	13	= =
	x(3)	0	0	= =
	x(4)	0	0	= =

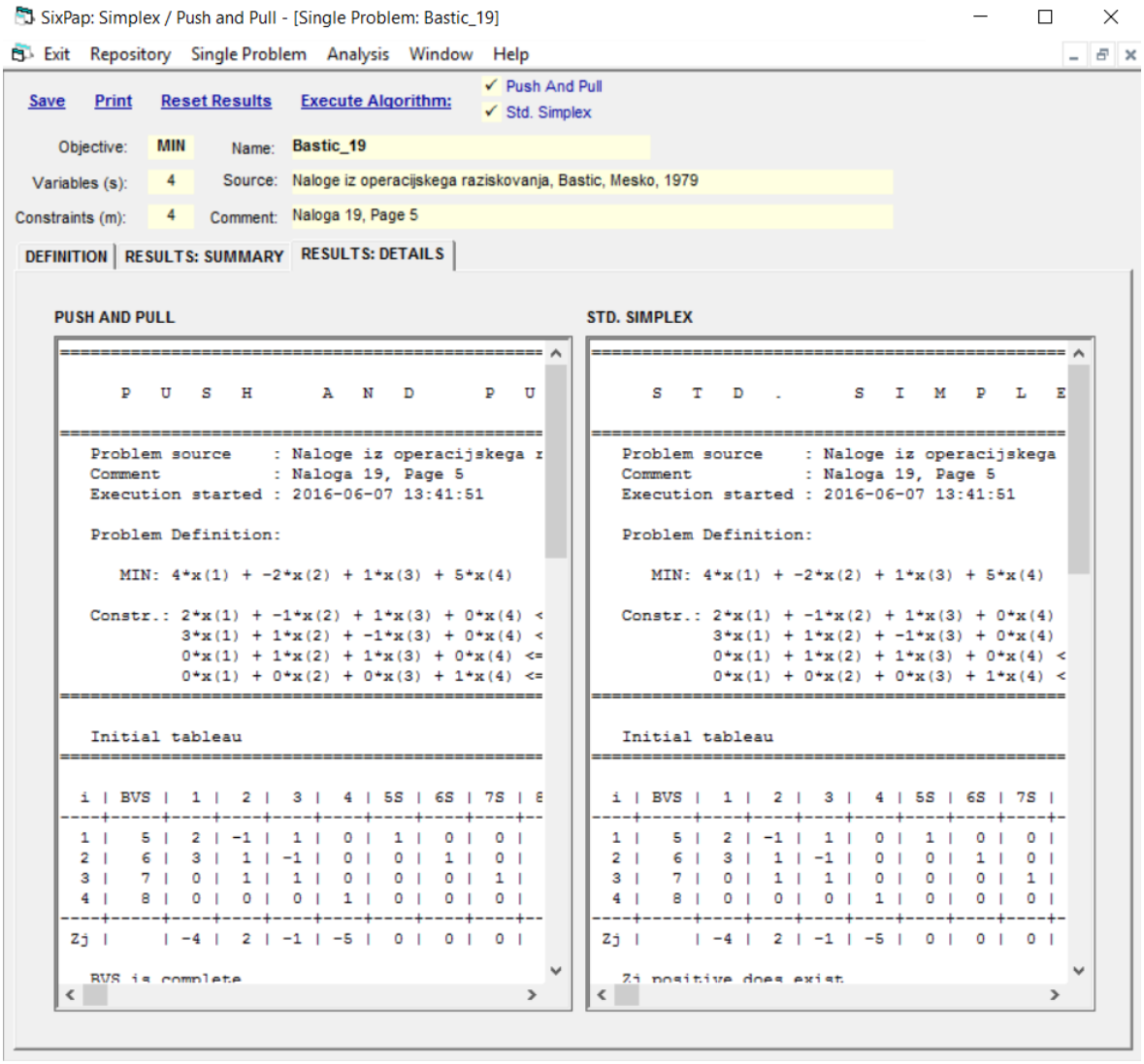
3.5.4.4 Področje s podrobnim potekom izvajanja algoritmov

Področje se nahaja na zavihku rezultati: podrobnosti (Slika 28, angl. *Results: Details*) in je prikazano v dveh vzporedno postavljenih oknih.

V vsakem od oken je za ustrezen algoritem prikazan podroben potek izvajanja, ki vsebuje v prvem delu definicijo problema, v nadaljevanju delne rezultate posameznih iteracij v tabelarični obliki, temu pa sledi ugotovitev naslednjega koraka in vzroki zanj.

Na koncu sledi prikaz podrobnih rezultatov izvajanja algoritma.

Slika 28: Ekranska slika okna posameznega problema – podrobnosti izvajanja



V tem področju lahko uporabnik po korakih vzporedno primerja delovanje enega in drugega algoritma ter ugotavlja razlike v poteku izvajanja.

Prikaz temelji na dveh usklajenih MyRichText kontrolah, ki omogočata širjenje ali krčenje vidnega polja ene kontrole in s tem obratno akcijo druge kontrole, kar še olajša primerjavo algoritmov po korakih. Vir prikaza kontrole je tekstovna datoteka, ki je nastajala ob izvajanju izbranega algoritma.

4 PRIMERJALNA ANALIZA

Namen programskega orodja SixPap je, poleg reševanja problemov linearnega programiranja, predvsem primerjava učinkovitosti obeh metod: standardne simpleks metode in metode potisni-povleci.

Merjenje učinkovitosti izvajanja deloma temelji na parametrih, ki so jih za primerjavo uporabili Arsham, Baloh, Damij in Grad (2003), dopolnjenih z merjenjem števila prehodov zank, števila odločitvenih stavkov ter z ugotavljanjem pojava degeneracije.

Primerjalna analiza je izvedena na naslednjih 15 primerih:

Primer 1.

Iščemo **maksimum** namenske funkcije: $2X_1 + 6X_2 + 8X_3 + 5X_4,$ (24)

ki zadošča naslednjim pogojem:

$$4X_1 + X_2 + 2X_3 + 2X_4 \geq 80$$
$$2X_1 + 5X_2 + 4X_4 \leq 40$$
$$2X_2 + 4X_3 + X_4 = 120$$
$$X_1, X_2, X_3, X_4 \geq 0.$$

Primer 2.

Iščemo **minimum** namenske funkcije: $2X_1 + X_2 + 8X_3 + 5X_4,$ (25)

ki zadošča naslednjim pogojem:

$$X_1 + 3X_2 + 2X_3 \leq 3$$
$$-2X_2 + X_3 \leq 1$$
$$X_1 - X_2 + X_3 = 2$$
$$X_1, X_2, X_3 \geq 0.$$

Primer 3.

Iščemo **maksimum** namenske funkcije: $20X_1 + 10X_2 + 40X_3 + 20X_4 + 15X_5,$ (26)

ki zadošča naslednjim pogojem:

$$X_1 + 4X_3 + 2X_4 + 4X_5 \leq 120$$
$$2X_1 + 5X_2 + 2X_3 + X_4 \leq 80$$
$$4X_1 + X_2 + 5X_3 + 4X_5 \leq 240$$
$$X_1, X_2, X_3, X_4, X_5 \geq 0.$$

Primer 4.

Iščemo **minimum** namenske funkcije: $X_1 + 3X_2 + 4X_3 + 10X_4,$ (27)

ki zadošča naslednjim pogojem:

$$X_1 + X_3 + X_4 \geq 10$$
$$X_2 + 2X_3 + 2X_4 \geq 25$$
$$X_1 + 2X_2 + X_5 \geq 20$$
$$X_1, X_2, X_3, X_4 \geq 0.$$

Primer 5.

Iščemo **maksimum** namenske funkcije: $100X_1 + 10X_2 + X_3,$ (28)

ki zadošča naslednjim pogojem:

$$X_1 \leq 1$$
$$20X_1 + X_2 \leq 100$$
$$200X_1 + 20X_2 + X_3 \leq 100000$$
$$X_1, X_2, X_3 \geq 0.$$

Primer 6.

Iščemo **minimum** namenske funkcije: $2X_1 + 2{,}5X_2$, (29)

ki zadošča naslednjim pogojem:

$$3X_1 + 4X_2 \geq 20000$$
$$0{,}75X_1 + 0{,}65X_2 \geq 4000$$
$$X_1, X_2 \geq 0.$$

Primer 7.

Iščemo **maksimum** namenske funkcije: $60X_1 + 70X_2 + 75X_3$, (30)

ki zadošča naslednjim pogojem:

$$3X_1 + 6X_2 + 4X_3 \leq 2400$$
$$5X_1 + 6X_2 + 7X_3 \leq 3600$$
$$3X_1 + 4X_2 + 5X_3 \leq 2600$$
$$X_1, X_2, X_3 \geq 0.$$

Primer 8.

Iščemo **maksimum** namenske funkcije: $2X_1 + 2{,}5X_2$, (31)

ki zadošča naslednjim pogojem:

$$3X_1 + 4X_2 \leq 20000$$
$$0{,}75X_1 + 0{,}65X_2 \leq 4000$$
$$X_1, X_2 \geq 0.$$

Primer 9.

Iščemo **maksimum** namenske funkcije: $12X_1 + 18X_2$, (32)

ki zadošča naslednjim pogojem:

$$2X_1 + 3X_2 \leq 33$$
$$X_1 + X_2 \leq 15$$
$$X_1 + 3X_2 \leq 27$$
$$X_1, X_2 \geq 0.$$

Primer 10.

Iščemo **minimum** namenske funkcije: $6X_1 - 3X_2 - 4X_3$, (33)

ki zadošča naslednjim pogojem:

$$-2X_1 + X_2 \leq 0$$
$$-3X_1 + X_2 + X_3 \geq 0$$
$$X_2 + X_3 = 24$$
$$X_1, X_2, X_3 \geq 0.$$

Primer 11.

Iščemo **maksimum** namenske funkcije: $2X_1 + X_2$, (34)

ki zadošča naslednjim pogojem:

$$X_1 + 3X_2 \leq 12$$
$$X_1 + 2X_2 \leq 10$$
$$2X_1 + 5X_2 \leq 30$$
$$X_1, X_2 \geq 0.$$

Primer 12.

Iščemo **minimum** namenske funkcije: $4X_1 - 2X_2 + X_3,$ (35)

ki zadošča naslednjim pogojem:

$$2X_1 - X_2 + X_3 \le 30$$
$$3X_1 + X_2 - X_3 \le 26$$
$$X_2 + X_3 \le 13$$
$$X_1, X_2, X_3 \ge 0.$$

Primer 13.

Iščemo **maksimum** namenske funkcije: $12X_1 + 6X_2 + 4X_3,$ (36)

ki zadošča naslednjim pogojem:

$$X_1 + 2X_2 \le 6$$
$$-X_1 - X_2 + 2X_3 \le 4$$
$$X_2 + X_3 \le 2$$
$$X_1, X_2, X_3 \ge 0.$$

Primer 14.

Iščemo **maksimum** namenske funkcije: $-X_1 + 21X_2 + 24X_3 + X_4 + 12X_5,$ (37)

ki zadošča naslednjim pogojem:

$$0.5X_1 + 6X_2 + 7X_3 - 2X_4 + X_5 \le 64$$
$$-X_1 + 2X_2 + 4X_3 + 2X_4 + 5X_5 \le 27$$
$$2X_1 + 3X_2 - X_3 - 4X_4 + 3X_5 \le 17$$
$$3X_1 + 2X_2 + 4X_3 - X_4 - X_5 \le 21$$
$$X_1, X_2, X_3, X_4, X_5 \ge 0.$$

Primer 15.

Iščemo **minimum** namenske funkcije: $3X_1 + X_2 - X_3,$ (38)

ki zadošča naslednjim pogojem:

$$2X_1 + X_2 - X_3 = 10$$
$$-X_1 + X_3 \le 6$$
$$3X_1 - 4X_3 \le 8$$
$$X_1, X_2, X_3 \ge 0.$$

Merjenje učinkovitosti in medsebojna primerjava obeh metod sta potekala s pomočjo štetja že omenjenih osnovnih računskih operacij, potrebnih za izračun optimalne rešitve.

Razultati primerjalne analize so prikazani v Tabelah 31, 32 in 33, pri čemer je prvi del vsem trem tabelam skupen in predstavlja karakteristike problema LP:

1. ime problema,
2. določitev, ali gre za iskanje maksimuma ali minimuma,
3. število pogojnih neenačb,
4. število spremenljivk namenske funkcije in
5. vrste relacij pogojnih neenačb.

Drugi del vseh treh tabel pa predstavlja podatke o rezultatu izračuna in osnovne kazalce učinkovitosti:

1. status rešitve,
2. indikator degeneracije,
3. optimalno vrednost namenske funkcije,
4. množico baznih vektorjev,
5. število iteracij,
6. število seštevanj/odštevanj,
7. število množenj/deljenj,
8. število prehodov preko programskih zank in
9. število prehodov preko odločitvenih stavkov.

Tabela 31 prikazuje podatke za izvajanje potisni-povleci metode, Tabela 32 standardne simpleks metode in Tabela 33 primerjavo med rezultati obeh metod ter razliko med vrednostmi kazalcev učinkovitosti.

Negativne vrednosti v Tabeli 33 gredo v korist metode potisni-povleci, pozitivne pa v korist standardne simpleks metode.

Tabela 31: Rezultati izvajanja potisni-povleci metode

Definition					**PUSH AND PULL**								
Name	**Obj.**	**m**	**s**	**Constr.**	**Status**	**Deg.**	**Value**	**BVS**	**Iter.**	**+/–**	***/÷**	**Loop**	**Dec.**
Example01	MAX	3	4	> < =	OPTIMAL	NO	280	1 5 3	4	92	119	234	285
Example02	MIN	4	3	< < = =	OPTIMAL	NO	–2,4583	2 5 3 1	3	79	94	199	233
Example03	MAX	3	5	< < <	OPTIMAL	NO	1,33E+03	3 1 8	2	62	82	180	210
Example04	MIN	3	4	> > >	OPTIMAL	NO	61	1 3 2	4	106	134	286	344
Example05	MAX	3	3	< < <	OPTIMAL	NO	100.000	4 5 3	7	160	193	351	416
Example06	MIN	2	2	> >	OPTIMAL	NO	1,29E+04	2 1	2	26	34	92	115
Example07	MAX	3	3	< < <	OPTIMAL	NO	43.200	4 1 6	2	50	61	144	173
Example08	MAX	2	2	< <	OPTIMAL	NO	1,29E+04	2 1	2	26	34	87	110
Example09	MAX	3	2	< < <	OPTIMAL	NO	198	1 4 2	2	44	52	116	142
Example10	MIN	3	3	< > =	OPTIMAL	YES	–96	5 3 1	3	62	76	169	201
Example11	MAX	3	2	< < <	OPTIMAL	NO	20	3 1 5	1	25	28	83	102
Example12	MIN	3	3	< < <	OPTIMAL	NO	–26	4 5 2	1	28	33	96	113
Example13	MAX	3	3	< < <	OPTIMAL	NO	80	1 5 3	2	50	59	140	163
Example14	MAX	4	5	< < < <	OPTIMAL	NO	219,0714	6 4 2 3	4	172	206	377	443
Example15	MIN	3	3	= < <	OPTIMAL	NO	10	2 4 5	1	23	29	95	103

Tabela 32: Rezultati izvajanja standardne simpleks metode

Definition					STD. SIMPLEX								
Name	**Obj.**	**m**	**s**	**Constr.**	**Status**	**Deg.**	**Value**	**BVS**	**Iter.**	**+/–**	***/÷**	**Loop**	**Dec.**
Example01	MAX	3	4	> < =	OPTIMAL	NO	280	1 5 3	4	256	318	316	300
Example02	MIN	4	3	< < = =	OPTIMAL	NO	–2,4583	2 5 3 1	3	237	274	284	245
Example03	MAX	3	5	< < <	OPTIMAL	NO	1,33E+03	3 1 8	2	143	178	191	175
Example04	MIN	3	4	> > >	OPTIMAL	YES	61	1 3 2	5	381	478	459	436
Example05	MAX	3	3	< < <	OPTIMAL	NO	100.000	4 5 3	7	328	404	402	400
Example06	MIN	2	2	> >	OPTIMAL	NO	1,29E+04	2 1	2	80	105	102	119
Example07	MAX	3	3	< < <	OPTIMAL	NO	43.200	4 1 6	2	113	137	147	141
Example08	MAX	2	2	< <	OPTIMAL	NO	1,29E+04	2 1	2	56	73	78	93
Example09	MAX	3	2	< < <	OPTIMAL	NO	198	1 4 2	2	98	118	125	127
Example10	MIN	3	3	< > =	OPTIMAL	YES	–96	5 3 1	3	181	221	224	217
Example11	MAX	3	2	< < <	OPTIMAL	NO	20	3 1 5	1	61	71	81	78
Example12	MIN	3	3	< < <	OPTIMAL	NO	–26	4 5 2	1	70	82	96	86
Example13	MAX	3	3	< < <	OPTIMAL	NO	80	1 5 3	2	113	135	147	137
Example14	MAX	4	5	< < < <	OPTIMAL	NO	219,0714	6 4 2 3	4	372	442	450	383
Example15	MIN	3	3	= < <	OPTIMAL	NO	10	2 4 5	4	200	245	249	245

Tabela 33: Primerjava rezultatov obeh metod

Definition					DIFFERENCE								
Name	**Obj.**	**m**	**s**	**Constr.**	**Status**	**Deg.**	**Value**	**BVS**	**Iter.**	**+/–**	***/÷**	**Loop**	**Dec.**
Example01	MAX	3	4	> < =	ok	ok	ok	ok	0	–164	–199	–82	–15
Example02	MIN	4	3	< < = =	ok	ok	ok	ok	0	–158	–180	–85	–12
Example03	MAX	3	5	< < <	ok	ok	ok	ok	0	–81	–96	–11	35
Example04	MIN	3	4	> > >	ok	DIFF	ok	ok	–1	–275	–344	–173	–92
Example05	MAX	3	3	< < <	ok	ok	ok	ok	0	–168	–211	–51	16
Example06	MIN	2	2	> >	ok	ok	ok	ok	0	–54	–71	–10	–4
Example07	MAX	3	3	< < <	ok	ok	ok	ok	0	–63	–76	–3	32
Example08	MAX	2	2	< <	ok	ok	ok	ok	0	–30	–39	9	17
Example09	MAX	3	2	< < <	ok	ok	ok	ok	0	–54	–66	–9	15
Example10	MIN	3	3	< > =	ok	ok	ok	ok	0	–119	–145	–55	–16
Example11	MAX	3	2	< < <	ok	ok	ok	ok	0	–36	–43	2	24
Example12	MIN	3	3	< < <	ok	ok	ok	ok	0	–42	–49	0	27
Example13	MAX	3	3	< < <	ok	ok	ok	ok	0	–63	–76	–7	26
Example14	MAX	4	5	< < < <	ok	ok	ok	ok	0	–200	–236	–73	60
Example15	MIN	3	3	= < <	ok	ok	ok	ok	–3	–177	–216	–154	–142

Iz Tabel 31, 32 in 33 je mogoče ugotoviti, da je metoda *potisni-povleci* v primerih 4 in 15 do rešitve prišla z manj iteracijami kot standardna simpleks metoda, v ostalih primerih pa je bilo število iteracij isto.

Glede na ostale kazalce učinkovitosti je možno ugotoviti, da metoda *potisni-povleci* uporablja manj osnovnih aritmetičnih operacij, vendar v povprečju več odločitvenih korakov.

SKLEP

V pričujočem delu sem najprej predstavil standardno simpleks metodo, ki velja za najbolj popularno metodo reševanja problemov linearnega programiranja, njeni začetki pa segajo v leto 1947 (Dantzig, 1968).

Poseben poudarek sem namenil razlagi algoritma, saj je razumevanje njegovega poteka ključno za uspešno računalniško implementacijo. Zato sem razlago podkrepil še z diagramom poteka in dvema zgledoma.

Temu sledi podroben prikaz računalniške implementacije algoritma, ki sem ga uporabil v programski rešitvi SixPap. Implementacijo sem pripravil v posebnem modulu, kar omogoča spremembe brez posega v osnovni program.

Pravila transformacije tabel po Gaussu so implementirana v posebnem modulu, saj jih v enaki meri uporabljata obe metodi, ki sta predmet tega dela. Računalniška implementacija je predstavljena v okviru poglavja o simpleks metodi, ker je to poglavje pred razlago metode *potisni-povleci*.

V nadaljevanju sem predstavil novejšo metodo z imenom potisni-povleci, ki jo je predstavil Arsham (1997). Metoda v osnovi temelji na simpleks metodi, a s to razliko, da ne uporablja t. i. umetnih spremenljivk in da ni treba, da množica baznih vektorjev začetne rešitve vsebuje vse elemente.

Tudi pri tej metodi sem se posvetil predvsem čim bolj nazorni predstavitvi algoritma, ki je, podobno kot simpleks metoda , za boljše razumevanje predstavljen z diagramom poteka in izvajanjem na dveh zgledih. Uporabljena zgleda sta ista kot pri simpleks metodi.

Tudi tukaj sledi podrobna predstavitev računalniške implementacije algoritma, ki je kot samostojni modul (v povezavi z modulom za Gaussovo transformacijo) uporabljen v programskem orodju SixPap.

V tretjem delu je predstavljeno orodje, ki sem ga razvil z upoštevanjem konceptov objektov orientiranega programiranja. Imenuje se SixPap in je, kot namiguje že ime, namenjeno reševanju problemov linearnega programiranja po eni, drugi ali obeh metodah. Poleg ostalega vsebuje tudi mehanizme za merjenje učinkovitosti izvajanja algoritmov in njuno primerjavo.

V zadnjem delu sem predstavil analizo učinkovitosti izvajanja obeh metod, ki je izvedena z orodjem SixPap na 15 primerih problemov linearnega programiranja.

Orodje SixPap sem razvil predvsem z namenom preverjanja učinkovitosti obeh metod, in sicer s štetjem iteracij, osnovnih računskih operacij, prehodov zank in odločitvenih stavkov. Štetje teh parametrov je z uporabo orodja na 15 primerih potrdilo Arshamovo hipotezo o večji učinkovitosti algoritma, ki izvira iz predpostavke, da se iskanje optimalne rešitve začne z rešitvijo, ki je precej bližje kot pri standardni simpleks metodi (Arsham, 1997).

Poleg merjenja učinkovitosti je v okviru orodja omogočeno hitro, enostavno in uporabniku prijazno reševanje problemov LP z obema metodama. Ker je v proces vključeno štetje operacij in beleženje podrobnega spremljanja izvajanja, je orodje v svoji osnovi namenjeno raziskovanju, saj je zaradi dodatnih obremenitev izvajanje počasnejše. To ne pomeni, da orodja ni mogoče uporabiti v praksi pri iskanju rešitev manj obsežnih problemov.

V orodju ni implementirano merjenje porabe procesorskega časa ali časa izvajanja posameznega algoritma, kar pomeni priložnost za nadgradnjo oziroma dopolnitev funkcionalnosti. Poleg tega je zaradi modularnosti mogoče dokaj enostavno razširiti uporabo še na druge, modificirane metode, kot je npr. izboljšana metoda potisni-povleci, ki jo predlaga Gao (2015).

LITERATURA IN VIRI

1. *AIMMS Linnear Programming*. Najdeno 15. junija 2016 na spletnem naslovu http://main.aimms.com/mathematical-programming/linear-programming
2. Arsham, H. (1997). Initialization of the Simplex Algorithm: An Artificial-Free Approach. *SIAM Review, 39*(4), 736-744.
3. Arsham, H., Baloh, P., Damij, T., & Grad, J. (2003). An algorithm for simplex tableau reduction with numerical comparison. *International journal of pure and applied mathematics, 4*(1), 57-85.
4. Arsham, H., Cimperman, G., Damij, N., Damij, T., & Grad, J. (2005). A computer implementation of the Push-and-Pull algorithm and its computational comparison with LP simplex method. *Applied mathematics and computation, 170*(1), 36-63.
5. Arsham, H., Damij, T., & Grad, J. (2003). An algorithm for simplex tableau reduction: the push-to-pull solution strategy. *Applied mathematics and computation, 137*(2-3), 525-547.
6. Bastič, M., & Meško, I. (1979). *Naloge iz operacijskega raziskovanja*. Maribor: Visoka ekonomsko komercialna šola.
7. Bazaraa, M., Jarvis, J., & Sherali, H. (1977). *Linear programming and Network Flows*. New York: Wiley.
8. Borgwardt, K. H. (1987). *The simplex method*. Berlin: Springer-Verlag.
9. Bradley, S., Hax, A., & Magnanti, T. (1977). *Applied Mathematical Programming. Reading*, MA: Addison Wesley.
10. Chvatal, V. (1983). *Linear programming*. New York: Freeman.
11. Curwin, J., Slater, R., & Eadson, D. (2013). *Quantitative methods for business decisions* (7th ed.). Andover: Cengage Learning.
12. Dantzig, G. B. (1968). *Linear programming and extensions*. Princeton, NJ: Princeton University Press.
13. Enge, A., & Huhn, P. (1998). A counterexample to H. Arsham's "initialization of the simplex algorithm: an artificial-free approach". SIAM Review, *40*(4), 1-6.
14. Feiring, B. R. (1986). *Linear programming : an introduction*. Beverly Hills: Sage.
15. Gao, P.-w. (2015). Improvement and its computer implementation of an artificial-free simplex-type algorithm by Arsham. *Applied Mathematics and Computation, 263*, 410–415.
16. Geary, R. C., & Spencer, J. E. (1973). *Elements of linear programming with economic aplications*. London: Griffin.
17. *GLPK - GNU Linear Programming Kit*. Najdeno 15. junija 2016 na spletnem naslovu https://www.gnu.org/software/glpk
18. Grasselli, J. (1986). Linearna algebra. V J. Grasselli, A. Vadnal & C. Velkovrh (ur.), *Linearna algebra. Linearno programiranje* (str. 9–134). Ljubljana: Društvo matematikov, fizikov in astronomov Slovenije.
19. Hadley, G. (1973). *Linear programming* (7th ed.). Reading: Addison-Wesley.

20. Heizer, J. H., & Render, B. (1993). *Production and operations management : strategies and tactics* (3rd ed.). Englewood Cliffs: Prentice Hall.
21. Hillier, F., & Lieberman, G. (1977). *Introduction to Mathematical Programming*. New York: McGraw-Hill.
22. Klee, V., & Minty, G. (1972). How Good is the Simplex Algorithm. V O. Shisha (ur.), *Inequalities-III* (str. 159–175, Academic Press.
23. *LINDO™ software products*. Najdeno 15. junija 2016 na spletnem naslovu http://www.lindo.com
24. *Linear Program Solver*. Najdeno 15. junija 2016 na spletnem naslovu https://sourceforge.net/projects/lipside/
25. Padberg, M. (1999). Linear *Optimization and Extensions* (2nd ed.). Berlin: Springer.
26. Ravindran, A., Ragsdell, K. M., & Reklaitis, G. V. (2006). *Engineering optimization : methods and applications*. Hoboken: John Wiley & Sons.
27. *The Push-and-Pull Solution Algorithm*. Najdeno 20. maja 2016 na spletnem naslovu http://home.ubalt.edu/ntsbarsh/Business-stat/opre/PartII.htm#rpushandpull
28. Vadnal, A. (1986). Linearno programiranje. V J. Grasselli, A. Vadnal & C. Velkovrh (ur.), *Linearna algrebra. Linearno programiranje* (str. 135-197). Ljubljana: Društvo matematikov, fizikov in astronomov Slovenije.
29. Vadnal, A. (1977). Rešeni *problemi linearnega programiranja*. Ljubljana: Državna založba Slovenije.
30. Winston, W. L. (2004). *Operations research : applications and algorithms*. Belmont: Brooks/Cole.

PRILOGE

KAZALO PRILOG

PRILOGA 1: Izpis kode modula Potisni-povleci

```
Attribute VB_Name = "ModulePushPull"

'   MODULE PUSHPULL: PUSH AND PULL METHOD IMPLEMENTATION

Private Problem As ClassProblem                 ' object holding problem
Private Result As ClassProblemResult            ' object holding results for Simplex
Private Inc As ClassCounters                    ' object holding counters
Private Definition As ClassProblemDefinition    ' object holding problem definition
Private noteDetail As ClassOutputDetailFile     ' object handling output
Private tableau() As Double                     ' tableau to work on
Private BVS() As Long                           ' basic variable set
Private D As Long                               ' number of slack/surplus variables
Private U As Long                               ' number of artificial variables
Private M As Long                               ' number of constraints
Private S As Long                               ' number of original variables
Private minMax As Integer                       ' objective indicator: MIN= -1 , MAX= 1
Private k As Long
Private r As Long
Private stepName As String
Private sortedCj As New ClassList
Private degeneracyOccured As Boolean

'
'   Push and Pull module entry point
'
Public Sub ExecutePushPull(onProblem As ClassProblem)

On Error GoTo errorExecutePushPull
    Set Problem = onProblem
    Set Result = onProblem.Result(PushPull)
    Set Inc = Result.Counters
    Set sortedCj.Inc = Inc
    Set Definition = onProblem.Definition
    Result.StartedWhen = Now
    Set noteDetail = Result.OutputDetailFile
    noteDetail.OpenOutputFile

    '   (A) Set dimensions and default values for matrix A, BVS, counters, ...
    InitiatePushPull
    '   (B) Convert problem into required form
    DoPreliminaries

    '   (1) Convert inequalities into equalities
    DoStep1

    '   (2) Construct initial tableau containing all slack variables in BVS
    DoStep2
    noteDetail.ResultDetailHeader
    noteDetail.SingleLine "Initial tableau", "=", 1, , "="
    noteDetail.PrintTableau tableau, S, D, U, M, BVS
    noteDetail.SingleLine "BVS is " & IIf(CompleteBVS, "", "NOT ") & "complete."

'---STEP 3: Generate a complete BVS
    Do While Not CompleteBVS
        '   rebuild sorted list based on last tableau row (zero values are excluded!)
        sortedCj.RebuildWith tableau

       '   look for new r and k within empty BVS-s
        DoStep3
```

```
        '   something odd happened: notify and exit method
        Inc.Decis
        If Result.Status <> NotCalculated Then GoTo exitDueToStatusChange
        Inc.Decis
        If sortedCj.IsEmpty Then

            '   k and r were not found, all possibilities were exhausted: notify and exit loop
            noteDetail.SingleLine "All possible incoming Xj have been used."
            Exit Do

        End If

        noteDetail.SingleLine "Iteration No.: " & Inc.Counter(cntIterations), "=", 1, , "="
        noteDetail.SingleLine "Step 3: completing BVS (k= " & k & ", r= " & r & ")"
        noteDetail.PrintTableau tableau, S, D, U, M, BVS
        Inc.Decis
        If degeneracyOccured Then
            noteDetail.SingleLine "DEGENERACY OCCURED."
            Result.Degeneracy = "YES"
        End If
        noteDetail.SingleLine "BVS is " & IIf(CompleteBVS, "", "NOT ") & "complete."

    Loop

'---END STEP 3

    '   loop to optimal solution
    noteDetail.SingleLine "Continue with Step 4."

    stepName = "4-PUSH"

    Do While True
        Inc.Loops

        Inc.Decis
        Select Case stepName

        Case "4-PUSH"

'-----------STEP 4: Push to optimal solution

            '   use sorted list, based on last row of tableau, to determine the largest element
            sortedCj.RebuildWith tableau

            '   do while positive element exists
            Do While sortedCj.Positive
                Inc.Loops

                '   calculate next solution
                DoStep4

                '   something odd happened: notify and exit method
                Inc.Decis
                If Result.Status <> NotCalculated Then GoTo exitDueToStatusChange

                noteDetail.SingleLine "Iteration No.: " & Inc.Counter(cntIterations), "=", 1, , "="
                noteDetail.SingleLine "Step 4: Push to optimal solution (k= " & k & ", r= " & r &
")"
                noteDetail.PrintTableau tableau, S, D, U, M, BVS
                Inc.Decis
                If degeneracyOccured Then
                    noteDetail.SingleLine "DEGENERACY OCCURED."
```

```
                    Result.Degeneracy = "YES"
                End If

                '   rebuild list and test for positive element
                sortedCj.RebuildWith tableau
                noteDetail.SingleLine "Zj positive does " & _
                                      IIf(sortedCj.Positive, "", "NOT ") & "exist."

            Loop

            '   continue with Step 5
            noteDetail.SingleLine "Continue with Step 5."
            stepName = "5-TEST"

'-----------END STEP 4

        Case "5-TEST"

'-----------STEP 5: Test the iteration results
            noteDetail.SingleLine "Step 5: Test the iteration results:", "=", 1
            noteDetail.SingleLine DoStep5(), , , 1

'-----------END STEP 5

        Case "6-PULL"

'-----------STEP 6:

            DoStep6

            '   something odd happened: notify and exit method
            Inc.Decis
            If Result.Status <> NotCalculated Then GoTo exitDueToStatusChange

            noteDetail.SingleLine "Iteration No.: " & Inc.Counter(cntIterations), "=", 1, , "="
            noteDetail.SingleLine "Step 6: Pull to feasible solution (k= " & k & ", r= " & r & ")"
            noteDetail.PrintTableau tableau, S, D, U, M, BVS
            Inc.Decis
            If degeneracyOccured Then
                noteDetail.SingleLine "DEGENERACY OCCURED."
                Result.Degeneracy = "YES"
            End If
            noteDetail.SingleLine "Continue with Step 5."

            stepName = "5-TEST"

'-----------END STEP 6

        Case "OPTIMAL"

            Result.Status = Optimal

            Exit Do

        End Select

    Loop

'---evaluate results
    ModuleGauss.CalculateOptimalSolution Result, Definition, tableau, BVS

    noteDetail.PrintTestResult
```

```
    noteDetail.PrintSummary

    noteDetail.CloseOutputFile

    Exit Sub

exitDueToStatusChange:

    '   something odd happened: notify and exit execution
    noteDetail.SingleLine "Status has changed: " & Result.StatusName
    noteDetail.SingleLine "Execution stopped.", , 1, 1, "="

    noteDetail.CloseOutputFile

    Exit Sub

errorExecutePushPull:

    '   error occured during method execution
    noteDetail.CloseOutputFile

    Err.Raise Err.Number, , "Execute Push and Pull method error." & _
                            NewLine(Err.Description)

    Exit Sub

End Sub

'
'   (A) Initiate Push And Pull: Set dimensions and default values
'
Private Sub InitiatePushPull()

    Dim i As Long, j As Long

    '   (a) initiate global method values
    stopExecuting = False
    Inc.Reset
    Result.Degeneracy = "NO"

    '   (b) read Objective, s and m
    Inc.Decis
    minMax = IIf(Definition.Objective = "MAX", -1, 1)
    M = Definition.M
    S = Definition.S

    '   (c) calculate D, U
    D = 0
    U = 0

    Inc.Loops M
    Inc.Decis M
    For i = 1 To M

        D = IIf(Definition.ConstraintType(i) <> "=", D + 1, D)
    Next i

    '   (d) dimension tableau, set initial values to Definition.A

    ReDim tableau(1 To M + 1, 1 To S + D + 1)

    Inc.Loops M * S
```

```
    For i = 1 To M

        For j = 1 To S

            tableau(i, j) = Definition.A(i, j)
        Next j
    Next i

    '(e) dimension BVS, set initial values to zero

    ReDim BVS(1 To M)

End Sub

'
'   (B) Preliminaries: Convert problem into required form
'
Private Sub DoPreliminaries()

    Dim i As Long, j As Long

    '   (a) The problem must be a maximization problem; set A(m+1) row

    Inc.Loops S
    Inc.MulDiv 2 * S
    For j = 1 To S

        tableau(M + 1, j) = -1 * minMax * Definition.Cj(j)
    Next j

    '   (b) RHS must be non-negative; set A(s+D+1) column
    '   "to avoid the occurence of a possible degeneracy, convert all inequality constr.
    '    with RHS=0 into 0 by multiplying each one of them by -1." (last article, page 5)

    For i = 1 To M
        Inc.Loops

        tableau(i, S + D + 1) = Abs(Definition.RHS(i))

        Inc.Decis
        If Definition.RHS(i) < 0 Then

            For j = 1 To S
                Inc.Loops

                tableau(i, j) = -1 * tableau(i, j)
                Inc.MulDiv
            Next j
        End If
    Next i

End Sub

'
'   (1) Step 1: Convert inequalities into equalities
'
Private Sub DoStep1()

    Dim i As Long, j As Long

    '   (a) Set tableau(i,j) value 1 or -1, regarding the relation type of the constraint
    j = S
```

```
    Inc.Loops M
    For i = 1 To M

        Inc.Decis
        If Definition.ConstraintType(i) <> "=" Then

            '   is slack or surplus

            j = j + 1
            Inc.AddSub

            '   slack or surplus, set tableau(i,j)
            Inc.Decis
            tableau(i, j) = IIf(Definition.ConstraintType(i) = "<", 1, -1)

            '   if RHS negative, multiply with -1
            Inc.Decis
            If (Definition.RHS(i) < 0) Then tableau(i, j) = -1 * tableau(i, j): Inc.MulDiv

        End If
    Next i

End Sub

'
'   (2) Step 2: Construct initial tableau containing all slack variables in BVS
'
Private Sub DoStep2()

    Dim i As Long, j As Long

    '   (a) Set tableau(i,j) value 1 or -1, regarding the relation type of the constraint

    j = S

    Inc.Loops M
    For i = 1 To M

        Inc.Decis
        If Definition.ConstraintType(i) <> "=" Then

            '   is slack or surplus

            j = j + 1: Inc.AddSub

            Inc.Decis
            If (Definition.ConstraintType(i) = "<") And _
               (Definition.RHS(i) >= 0) Then

                '   is slack and RHS was non-negative

                BVS(i) = j

            End If
        End If
    Next i

    '   execution has just begun => status should be NotCalculated

    Result.Status = NotCalculated

End Sub
```

```
'
'   (3) Step 3: Generate a complete BVS
'
Private Sub DoStep3()

    Dim modTableau()
    Dim i As Long, j As Long

    '   loop while k and r are found or sorted list is empty

    Do While Not sortedCj.IsEmpty
        Inc.Loops
        '   (1) Incoming variable is the one with the largest Cj

        k = sortedCj.Largest.j

        '   (2) Choose the smallest !!positive!! C/R if possible

        '   create modified tableau (clear rows with occupied bvs)
        modTableau = ModuleUtilities.CloneMatrix(tableau)
        For i = 1 To M
            If BVS(i) <> 0 Then
                For j = 1 To S
                    modTableau(i, j) = 0
                Next j
            End If
        Next i

        r = ModuleGauss.ColumnRatioTest(Inc, modTableau, k, degeneracyOccured, True)


        Inc.Decis
        If r = 0 Then

            '   column ratio test didn't give any results, try with next biggest element in sorted
list

            sortedCj.RemoveLargest

        Else

            '   (3a) Calculated r-row in BVS is empty, let's pivot
            Inc.Decis
            If BVS(r) = 0 Then

                '   increase iteration count and check for boundaries (to prevent infinite loop)
                Inc.Decis
                If ModuleGauss.IterationCountExceedsMaxAllowed(Inc) Then

                    '   too many iterations: change status and leave this loop (to prevent infinite
loop)

                    Result.Status = MaxIterationsExceeded
                    Exit Sub

                End If

                '   success: k and r are found, execute tableau pivoting and leave this loop

                ModuleGauss.GaussPivot Inc, tableau, k, r, BVS
                Exit Do
```

```
            Else

                '   (3b) BVS(r) is already occupied, repeat (1) with next largest Cj

                sortedCj.RemoveLargest

            End If
        End If
    Loop
End Sub

'
'   (4) Step 4: Push to optimal solution
'
Private Sub DoStep4()

    '   (1) Incoming variable is the one with the largest Cj

    k = sortedCj.Largest.j

    '   (2) Choose the smallest !!positive!! C/R if possible

    r = ModuleGauss.ColumnRatioTest(Inc, tableau, k, degeneracyOccured)

    Inc.Decis
    If r = 0 Then

        '   column ratio test didn't give any result => is it infesible or unbounded?

        Result.Status = Unbounded
        Exit Sub

    End If

    Inc.Decis
    If ModuleGauss.IterationCountExceedsMaxAllowed(Inc) Then

        '   too many iterations: change status and leave this loop (to prevent infinite loop)

        Result.Status = MaxIterationsExceeded
        Exit Sub

    End If

    '   success: calculate new tableau

    ModuleGauss.GaussPivot Inc, tableau, k, r, BVS

End Sub

'
'   (5) Step 5: Testing
'
Private Function DoStep5() As String

    Dim decision As String

    '   use sorted list to determine existance of positive Cj

    sortedCj.RebuildWith tableau
```

```
    '   decide which step goes next

    decision = ""
    decision = decision & IIf(negativeRHS, "F", "T"): Inc.Decis
    decision = decision & IIf(sortedCj.Positive, "F", "T"): Inc.Decis

    Inc.Decis
    Select Case decision

    Case "TT"

        '   All RHS >= 0 and all Cj <= 0 : solution is optimal

        stepName = "OPTIMAL"
        DoStep5 = "All RHS >= 0 and all Zj <= 0 : solution is optimal"

    Case "TF", "FF"

        '   Not all Cj <= 0: continue with PUSH (Step 4)

        stepName = "4-PUSH"
        DoStep5 = "Not all Zj <= 0: continue with PUSH (Step 4)"

    Case "FT"

        '   Not all RHS >= 0: continue with PULL (Step 6)

        stepName = "6-PULL"
        DoStep5 = "Not all RHS >= 0: continue with PULL (Step 6)"

    End Select

End Function

'
'   (6) Step 6: Pull to feasible solution
'
Private Sub DoStep6()

    '   (a) Use the dual simplex pivoting rules to determine k and r (R/R test)

    ModuleGauss.RowRatioTest Inc, tableau, k, r

    Inc.Decis
    If r = 0 Then

        '   row ratio test didn't give any result => is it infesible or unbounded?

        Result.Status = Infeasible
        Exit Sub

    End If

    Inc.Decis
    If ModuleGauss.IterationCountExceedsMaxAllowed(Inc) Then

        '   too many iterations: change status and leave this loop (to prevent infinite loop)

        Result.Status = MaxIterationsExceeded
        Exit Sub

    End If
```

```
    '   (b) generate the next tableau

    ModuleGauss.GaussPivot Inc, tableau, k, r, BVS

End Sub

'
'   check if BVS is complete
'
Private Function CompleteBVS() As Boolean

    Dim i As Long

    CompleteBVS = True

    For i = 1 To M
        Inc.Loops

        Inc.Decis
        If BVS(i) = 0 Then

            CompleteBVS = False

            Exit Function

        End If
    Next i
End Function

'
'   check if RHS negative exists (pull phase)
'
Private Function negativeRHS() As Boolean
    Dim i As Long
    negativeRHS = False
    For i = 1 To M
        Inc.Loops
        Inc.Decis
        If tableau(i, S + D + 1) < 0 Then
            negativeRHS = True
            Exit Function
        End If
     Next i
End Function
```

PRILOGA 2: Izpis kode modula Simpleks

```
Attribute VB_Name = "ModuleSimplex"

'   MODULE SIMPLEX: STD. SIMPLEX METHOD IMPLEMENTATION

Option Explicit

Private Problem As ClassProblem                 ' object holding problem
Private Result As ClassProblemResult            ' object holding results for Simplex
Private Inc As ClassCounters                    ' object holding counters
Private Definition As ClassProblemDefinition    ' object holding problem definition
Private noteDetail As ClassOutputDetailFile     ' object handling output
Private tableau() As Double                     ' tableau to work on
Private BVS() As Long                           ' basic variable set
Private D As Long                               ' number of slack/surplus variables
Private U As Long                               ' number of artificial variables
Private M As Long                               ' number of constraints
Private S As Long                               ' number of original variables
Private minMax As Integer                       ' objective indicator: MIN= -1 , MAX= 1
Private bigM As Double                          ' the big M value
Private k As Long
Private r As Long
Private stopExecuting As Boolean                ' execution flag
Private simplexCj() As Double                   ' last row in tableau - why do I need that ??
Private existsZjPositive As Boolean             ' positive Zj - Cj existance flag
Private biggestZjColumnIndex As Long            ' biggest Zj - Cj column index (next possible k)
Private degeneracyOccured As Boolean

'
'   entry point for standardSimplex method execution
'
Public Sub ExecuteSimplex(onProblem As ClassProblem)

On Error GoTo errorExecuteSimplex

    Set Problem = onProblem
    Set Result = onProblem.Result(Simplex)
    Set Inc = Result.Counters
    Set Definition = onProblem.Definition

    Result.StartedWhen = Now


    Set noteDetail = Result.OutputDetailFile
    noteDetail.OpenOutputFile

    '   (A) set dimensions and default values for matrix A, BVS, counters, ...
    InitiateSimplex

    '   (B) construct initial tableau
    ConstructInitialTableau

    noteDetail.ResultDetailHeader
    noteDetail.SingleLine "Initial tableau", "=", 1, , "="
    noteDetail.PrintTableau tableau, S, D, U, M, BVS
    noteDetail.SingleLine "Zj positive does " & IIf(existsZjPositive, "", "NOT ") & "exist."

    '   loop to optimal solution
    Do While stopExecuting = False
        Inc.Loops
```

```
        '   (1) Calculate new tableau
        CalculateNewTableau

        noteDetail.SingleLine "Iteration No.: " & Inc.Counter(cntIterations), "=", 1, , "="
        noteDetail.SingleLine "k= " & k & ", r= " & r
        noteDetail.PrintTableau tableau, S, D, U, M, BVS
        Inc.Decis
        If degeneracyOccured Then
            noteDetail.SingleLine "DEGENERACY OCCURED."
            Result.Degeneracy = "YES"
        End If
        noteDetail.SingleLine "Zj positive does " & IIf(existsZjPositive, "", "NOT ") & "exist."

    Loop

'---evaluate result:
    Select Case Result.Status

    Case EnumStatus.Optimal

        ModuleGauss.CalculateOptimalSolution Result, Definition, tableau, BVS

        noteDetail.PrintTestResult
        noteDetail.PrintSummary

    Case Else

        noteDetail.SingleLine "Status has changed: " & Result.StatusName
        noteDetail.SingleLine "Execution stopped.", , 1, 1, "="

    End Select

    noteDetail.CloseOutputFile

    Exit Sub

errorExecuteSimplex:

    noteDetail.CloseOutputFile

    Err.Raise Err.Number, , "Execute Std. Simplex method error." & _
                            NewLine(Err.Description)

End Sub

'
'   (A) set dimensions and default values
'
Private Sub InitiateSimplex()

    Dim i As Long, j As Long

    '   (a) initiate global method values
    stopExecuting = False
    Inc.Reset

    existsZjPositive = False
    biggestZjColumnIndex = 0

    '   (b) read Objective, s and m
    minMax = IIf(Definition.Objective = "MAX", -1, 1): Inc.Decis
    M = Definition.M
```

```
    S = Definition.S

    '   (c) calculate D and U
    D = 0
    U = 0

    Inc.Loops M
    For i = 1 To M

        D = IIf(Definition.ConstraintType(i) <> "=", D + 1, D): Inc.Decis
        U = IIf(Definition.ConstraintType(i) <> "<", U + 1, U): Inc.Decis

    Next i

    '   (d) dimension tableau, set initial values to A or zero
    ReDim tableau(1 To M + 1, 1 To S + D + U + 1)

    Inc.Loops M * S
    For i = 1 To M

        For j = 1 To S

            tableau(i, j) = Definition.A(i, j)

        Next j

    Next i

    '   (e) dimension BVS, set initial values to zero
    ReDim BVS(1 To M)

    '   (f) calculate BIG M
    bigM = 0

    Inc.Loops S
    Inc.Decis S
    For i = 1 To S

        bigM = IIf(Definition.Cj(i) > bigM, Definition.Cj(i), bigM)

    Next i

    bigM = 10 * bigM: Inc.MulDiv

    '   (g) dimension simplexCj (expand original Cj with surplus/slack and artificial variables)
    ReDim simplexCj(S + D + U + 1)

    Inc.Loops S
    For j = 1 To S

        simplexCj(j) = Definition.Cj(j):

    Next j

End Sub

'
'   (B) Construct initial tableau for Simplex
'
Private Sub ConstructInitialTableau()

    Dim i As Long
```

```
    Dim Dj As Long: Dj = 0
    Dim Uj As Long: Uj = 0

    '   (a) create slack, surplus and artificial
    Inc.Loops M
    Inc.Decis M
    For i = 1 To M

        Select Case Definition.ConstraintType(i)

        Case "="

            Uj = Uj + 1
            tableau(i, S + D + Uj) = 1
            simplexCj(S + D + Uj) = minMax * bigM
            BVS(i) = S + D + Uj
            Inc.MulDiv
            Inc.AddSub 3

        Case "<"

            Dj = Dj + 1
            tableau(i, S + Dj) = 1
            BVS(i) = S + Dj
            Inc.AddSub 2

        Case ">"

            Uj = Uj + 1
            Dj = Dj + 1
            tableau(i, S + Dj) = -1
            simplexCj(S + D + Uj) = minMax * bigM
            tableau(i, S + D + Uj) = 1
            BVS(i) = S + D + Uj
            Inc.AddSub 4
            Inc.MulDiv

        End Select

    Next i

    '   (b) read RHS
    Inc.Loops M
    For i = 1 To M

        tableau(i, S + D + U + 1) = Definition.RHS(i)

    Next i

    '   (c) calculate z(j)-c(j)
    CalculateZ

    '   if Zj positive doesn't exist stop executing
    stopExecuting = Not existsZjPositive

End Sub

'
'   calculate new incoming and outgoing vector, pivot tableau
'
Private Sub CalculateNewTableau()
```

```
    '   find best k
    k = biggestZjColumnIndex

    Result.Status = NotCalculated

    stopExecuting = True

    '   calculate r with C/R test
    r = ModuleGauss.ColumnRatioTest(Inc, tableau, k, degeneracyOccured)

    '   what if column ratio test doesn't give a result
    Inc.Decis
    If r = 0 Then

        Result.Status = EnumStatus.Unbounded
    End If

    '   increase iteration count and check for boundaries
    Inc.Decis
    If ModuleGauss.IterationCountExceedsMaxAllowed(Inc) Then

        Result.Status = MaxIterationsExceeded
    End If

    '   something went wrong
    Inc.Decis
    If Result.Status <> NotCalculated Then Exit Sub

    ModuleGauss.GaussPivot Inc, tableau, k, r, BVS

    CalculateZ

    '   check for positive Zj
    stopExecuting = Not existsZjPositive

End Sub

'
'   calculating last row; alter for maximum
'
Private Function CalculateZ()

    Dim i As Long
    Dim j As Long
    Dim tempZ As Double
    Dim tempBiggestZj As Double: tempBiggestZj = 0

    existsZjPositive = False
    biggestZjColumnIndex = 0

    '   (a) calculate last row
    For j = 1 To S + D + U + 1

        tempZ = 0

        Inc.Loops M
        For i = 1 To M

            '   calculate cumulative value
            tempZ = tempZ + simplexCj(BVS(i)) * tableau(i, j)
            Inc.AddSub
            Inc.MulDiv
```

```
        Next i

        Inc.Decis
        If j < S + D + U + 1 Then

            '   calculate last row in tableau
            tableau(M + 1, j) = minMax * (tempZ - simplexCj(j))
            Inc.MulDiv

            '   check for positive value
            existsZjPositive = IIf(tableau(M + 1, j) > ModuleGauss.eta, True, existsZjPositive)
            Inc.Decis

            '   check for biggest value
            Inc.Decis
            If tableau(M + 1, j) > tempBiggestZj Then

                tempBiggestZj = tableau(M + 1, j)
                biggestZjColumnIndex = j

            End If

        Else

            '   current objective function value
            tableau(M + 1, j) = tempZ

        End If

    Next j

    Inc.Decis
    If Not existsZjPositive Then Result.Status = Optimal

End Function
```

PRILOGA 3: Izpis kode modula Gauss

```
Attribute VB_Name = "ModuleGauss"

' GAUSS PIVOTING AND OTHER SUPPORT FUNCTIONS MODULE

Public Const eta As Double = 0.0000001  ' error margine

Private Const maximumIterationsAllowed = 100    ' safety reasons

' pivot given tableau using gauss pivoting rules

Public Sub GaussPivot(Inc As ClassCounters, tableau, _
                      k As Long, _
                      r As Long, _
                      BVS)

    Dim i As Long
    Dim j As Long
    Dim keyA As Double

    ' key coefficient for pivoting
    keyA = tableau(r, k)

    ' new Basic Vector
    BVS(r) = k

    ' row r
    Inc.Loops UBound(tableau, 2) - LBound(tableau, 2) + 1
    Inc.MulDiv UBound(tableau, 2) - LBound(tableau, 2) + 1
    For j = LBound(tableau, 2) To UBound(tableau, 2)

        tableau(r, j) = tableau(r, j) / keyA

    Next j

    ' rows other than r
    For i = LBound(tableau, 1) To UBound(tableau, 1)

        Inc.Decis
        If i = r Then i = i + 1: Inc.AddSub

        For j = LBound(tableau, 2) To UBound(tableau, 2)

            Inc.Decis
            If j = k Then j = j + 1: Inc.AddSub

            tableau(i, j) = tableau(i, j) - tableau(i, k) * tableau(r, j)
            Inc.MulDiv
            Inc.AddSub

            Inc.Loops
        Next j

        tableau(i, k) = 0

    Next i

End Sub

' column ratio test
```

```
Public Function ColumnRatioTest(Inc As ClassCounters, tableau, _
                                k As Long, _
                                ByRef degeneracyOccured As Boolean, _
                                Optional ignoreDegeneracy As Boolean = False) As Long

    Dim minimalColumnRatio As Double: minimalColumnRatio = -1
    Dim tempColumnRatio As Double

    Dim i As Long, j As Long

    Dim potencialRs() As Long

    Dim tempR As Long: tempR = 0

    Dim tempDegeneracy As Boolean: tempDegeneracy = False

    ReDim potencialRs(1 To 1)
    potencialRs(1) = 0

    ColumnRatioTest = 0

    For i = 1 To UBound(tableau, 1) - 1 '= m

        ' both RHS(k) and A(i,k) MUST be positive ?????
        Inc.Decis
        If ((tableau(i, k) > 0) And (tableau(i, UBound(tableau, 2)) >= 0)) Then
'       If ((tableau(i, k) > 0) And (tableau(i, UBound(tableau, 2)) > 0)) Then

            tempColumnRatio = tableau(i, UBound(tableau, 2)) / tableau(i, k)
            Inc.MulDiv

            ' posible degeneracy occurs
            Inc.Decis
            If tempColumnRatio = minimalColumnRatio Then

                tempDegeneracy = True

                ReDim Preserve potencialRs(1 To UBound(potencialRs) + 1)
                potencialRs(UBound(potencialRs)) = i

            End If

            Inc.Decis
            If ((tempColumnRatio < minimalColumnRatio) Or (minimalColumnRatio = -1)) Then

                tempDegeneracy = False

                ReDim potencialRs(1 To 1)
                potencialRs(1) = i

                minimalColumnRatio = tempColumnRatio

            End If
        End If
    Next i
    Inc.Loops UBound(tableau, 1)

    '   degeneracy handling off: break ties arbitrarily (last in)
    If ignoreDegeneracy Then

        degeneracyOccured = False
```

```
        ColumnRatioTest = potencialRs(1)
        Exit Function
    End If


    ' handle possible degeneracy
    Inc.Decis
    If tempDegeneracy Then

        j = 0

        Do While tempR = 0

            j = j + 1
            Inc.AddSub

            ' skip k - column
            Inc.Decis
            If j = k Then j = j + 1: Inc.AddSub

            ' check boundaries of tableau
            Inc.Decis
            If j > UBound(tableau, 2) - 1 Then Exit Do

            minimalColumnRatio = -1

            For i = LBound(potencialRs) To UBound(potencialRs)
                Inc.Loops

                Inc.Decis
                If ((tableau(potencialRs(i), k) > 0) And (tableau(potencialRs(i), j)) >= 0) Then
'               If ((tableau(potencialRs(i), k) > 0) And (tableau(potencialRs(i), j)) > 0) Then

                    tempColumnRatio = tableau(potencialRs(i), j) / tableau(potencialRs(i), k)
                    Inc.MulDiv

                    If tempColumnRatio = minimalColumnRatio Then tempR = 0

                    Inc.Decis
                    If ((tempColumnRatio < minimalColumnRatio) Or (minimalColumnRatio = -1)) Then

                        tempR = potencialRs(i)

                        minimalColumnRatio = tempColumnRatio

                    End If
                End If
            Next i
        Loop
    Else

        tempR = potencialRs(1)
    End If

    degeneracyOccured = tempDegeneracy

    ColumnRatioTest = tempR

End Function

' row ratio test (dual simplex)
```

```
Public Sub RowRatioTest(Inc As ClassCounters, tableau, _
                        ByRef k As Long, _
                        ByRef r As Long)

    Dim i As Long, j As Long

    Dim minimalRHS As Double: minimalRHS = 0

    Dim maximalRowRatio As Double
    Dim tempRowRatio As Double

    Dim tempR() As Long

    k = 0
    r = 0

    ' build potencial r-values (theta)
    For i = 1 To UBound(tableau, 1) - 1 '=m
        Inc.Loops

        ' RHS is negative
        Inc.Decis
        If tableau(i, UBound(tableau, 2)) < 0 Then

            ' min RHS already exists
            Inc.Decis
            If minimalRHS = tableau(i, UBound(tableau, 2)) Then

                ReDim Preserve tempR(UBound(tempR) + 1)
                tempR(UBound(tempR)) = i

            End If

            ' new min RHS
            Inc.Decis
            If tableau(i, UBound(tableau, 2)) < minimalRHS Then

                ReDim tempR(0)
                tempR(0) = i

                minimalRHS = tableau(i, UBound(tableau, 2))

            End If
        End If
    Next i

    ' find best RR value

    maximalRowRatio = 0

    For i = LBound(tempR) To UBound(tempR)

        For j = 1 To UBound(tableau, 2) - 1
            Inc.Loops

            Inc.Decis
            If tableau(UBound(tableau, 1), j) < 0 Then  ' Cj has to be negative

                Inc.Decis
                If tableau(tempR(i), j) < 0 Then ' A(i,j) has to be negative

                    tempRowRatio = tableau(UBound(tableau, 1), j) / tableau(tempR(i), j)
```

```
                    Inc.MulDiv

                    Inc.Decis
                    If maximalRowRatio < tempRowRatio Then

                        maximalRowRatio = tempRowRatio
                        r = tempR(i)
                        k = j
                    End If
                End If
            End If
        Next j
    Next i
End Sub

' increase iteration count and check if too big

Public Function IterationCountExceedsMaxAllowed(Inc As ClassCounters) As Boolean

    Inc.Iters

    IterationCountExceedsMaxAllowed = (Inc.Counter(cntIterations) > maximumIterationsAllowed)
End Function



' calculate optimal solution

Public Sub CalculateOptimalSolution(Result As ClassProblemResult, _
                                    Definition As ClassProblemDefinition, _
                                    tableau, _
                                    BVS)

    Dim i As Long, j As Long
    Dim trivialSolution As Boolean: trivialSolution = True

    ' calculate values
    For j = 1 To Definition.S

        Result.Xj(j) = 0

    Next j

    Result.FunctionValue = 0

    For i = 1 To Definition.M

        Result.BVS(i) = BVS(i)

        If (BVS(i) > 0) And (BVS(i) <= Definition.S) Then

            Result.Xj(BVS(i)) = tableau(i, UBound(tableau, 2))

            trivialSolution = IIf(Result.Xj(BVS(i)) <> 0, False, trivialSolution)

            Result.FunctionValue = Result.FunctionValue + _
                Result.Xj(BVS(i)) * Definition.Cj(BVS(i))

        End If

    Next i
```

```
    Result.TestResult = "FAILED"

    '   testing for conditions
    Dim tempValue As Double
    Dim tempTest As Boolean: tempTest = True
    For i = 1 To Definition.M

        tempValue = 0

        For j = 1 To Definition.S

            tempValue = tempValue + Definition.A(i, j) * Result.Xj(j)

        Next j

        Result.ConstraintTest(i) = False

        Select Case Definition.ConstraintType(i)

        Case "="

            If Abs(tempValue - Definition.RHS(i)) <= eta Then

                Result.ConstraintTest(i) = True

            End If

        Case ">"

            If tempValue >= Definition.RHS(i) - eta Then

                Result.ConstraintTest(i) = True

            End If

        Case "<"

            If tempValue <= Definition.RHS(i) + eta Then

                Result.ConstraintTest(i) = True

            End If

        End Select

        tempTest = (tempTest And Result.ConstraintTest(i))

    Next i

    Result.TestResult = IIf(tempTest, "OK", "FAILED")

    '   recalculate status
    If trivialSolution Then Result.Status = Trivial
    If Result.TestResult = "FAILED" Then Result.Status = TestFailed

End Sub
```

PRILOGA 4: Primer izpisa definicijske datoteke s povzetkom rezultatov

```
name: Bastic_19
source: Naloge iz operacijskega raziskovanja, Bastic, Mesko, 1979
comment: Naloga 19, Page 5
--------------------------------------------------
objective: MIN
m: 4
s: 4
C(j): 4 -2 1 5
constraints:
(1): 2 -1 1 0 < 30
(2): 3 1 -1 0 < 26
(3): 0 1 1 0 < 13
(4): 0 0 0 1 < 20
--------------------------------------------------
method: PUSH AND PULL
status: OPTIMAL
value: -26
bvs(i): 5 6 2 8
x(j): 0 13 0 0
degeneracy: NO
iterations: 1
adds/subs: 45
mults/divs: 51
loops: 144
decisions: 162
testing: OK
started: 2016-06-07 13:41:51
detail file: Bastic_19.pap
--------------------------------------------------
method: STD. SIMPLEX
status: OPTIMAL
value: -26
bvs(i): 5 6 2 8
x(j): 0 13 0 0
degeneracy: NO
iterations: 1
adds/subs: 117
mults/divs: 132
loops: 155
decisions: 118
testing: OK
started: 2016-06-07 13:41:51
detail file: Bastic_19.six
--------------------------------------------------
```

PRILOGA 5: Primer izpisa podrobnega poteka algoritma potisni-povleci

```
================================================================================================
=

      P   U   S   H       A   N   D       P   U   L   L

================================================================================================
=
   Problem source    : Naloge iz operacijskega raziskovanja, Bastic, Mesko, 1979
   Comment           : Naloga 19, Page 5
   Execution started : 2016-06-07 13:41:51

   Problem Definition:

      MIN: 4*x(1) + -2*x(2) + 1*x(3) + 5*x(4)

   Constr.: 2*x(1) + -1*x(2) + 1*x(3) + 0*x(4) <= 30
            3*x(1) + 1*x(2) + -1*x(3) + 0*x(4) <= 26
            0*x(1) + 1*x(2) + 1*x(3) + 0*x(4) <= 13
            0*x(1) + 0*x(2) + 0*x(3) + 1*x(4) <= 20
================================================================================================
=

   Initial tableau
================================================================================================
=

  i | BVS |  1 |  2 |  3 |  4 | 5S | 6S | 7S | 8S | RHS
----+-----+----+----+----+----+----+----+----+----+-----
  1 |   5 |  2 | -1 |  1 |  0 |  1 |  0 |  0 |  0 |  30
  2 |   6 |  3 |  1 | -1 |  0 |  0 |  1 |  0 |  0 |  26
  3 |   7 |  0 |  1 |  1 |  0 |  0 |  0 |  1 |  0 |  13
  4 |   8 |  0 |  0 |  0 |  1 |  0 |  0 |  0 |  1 |  20
----+-----+----+----+----+----+----+----+----+----+-----
 Zj |     | -4 |  2 | -1 | -5 |  0 |  0 |  0 |  0 |   0

   BVS is complete.
   Continue with Step 4.
================================================================================================
=

   Iteration No.: 1
================================================================================================
=
   Step 4: Push to optimal solution (k= 2, r= 3)

  i | BVS |  1 | 2 |  3 |  4 | 5S | 6S | 7S | 8S | RHS
----+-----+----+---+----+----+----+----+----+----+-----
  1 |   5 |  2 | 0 |  2 |  0 |  1 |  0 |  1 |  0 |  43
  2 |   6 |  3 | 0 | -2 |  0 |  0 |  1 | -1 |  0 |  13
  3 |   2 |  0 | 1 |  1 |  0 |  0 |  0 |  1 |  0 |  13
  4 |   8 |  0 | 0 |  0 |  1 |  0 |  0 |  0 |  1 |  20
----+-----+----+---+----+----+----+----+----+----+-----
 Zj |     | -4 | 0 | -3 | -5 |  0 |  0 | -2 |  0 | -26

   Zj positive does NOT exist.
   Continue with Step 5.
================================================================================================
=
```

```
   Step 5: Test the iteration results:
   All RHS >= 0 and all Zj <= 0 : solution is optimal

=============================================================================================================
=

      R E S U L T   T E S T I N G
=============================================================================================================
=

   constraint no.   1 TRUE : -13 <= 30
   constraint no.   2 TRUE : 13 <= 26
   constraint no.   3 TRUE : 13 <= 13
   constraint no.   4 TRUE : 0 <= 20

   Test result: OK

=============================================================================================================
=

      S U M M A R Y
=============================================================================================================
=

   Status: OPTIMAL

   Degeneracy: NO

   Test to constraints: OK

   MIN: -26

   BVS: [5, 6, 2, 8 ]

   x(1) = 0
   x(2) = 13
   x(3) = 0
   x(4) = 0

   Number of iterations: 1
   Number of additions/subtractions: 45
   Number of multiplications/divisions: 51
   Number of loops: 144
   Number of decisions: 162



=============================================================================================================
=
```

PRILOGA 6: Primer izpisa podrobnega poteka algoritma simpleks

```
=============================================================================================================
=

      S   T   D   .       S   I   M   P   L   E   X

=============================================================================================================
=
   Problem source    : Naloge iz operacijskega raziskovanja, Bastic, Mesko, 1979
   Comment           : Naloga 19, Page 5
   Execution started : 2016-06-07 13:41:51

   Problem Definition:

      MIN: 4*x(1) + -2*x(2) + 1*x(3) + 5*x(4)

   Constr.: 2*x(1) + -1*x(2) + 1*x(3) + 0*x(4) <= 30
            3*x(1) + 1*x(2) + -1*x(3) + 0*x(4) <= 26
            0*x(1) + 1*x(2) + 1*x(3) + 0*x(4) <= 13
            0*x(1) + 0*x(2) + 0*x(3) + 1*x(4) <= 20
=============================================================================================================
=

   Initial tableau
=============================================================================================================
=

  i | BVS |  1 |  2 |  3 |  4 | 5S | 6S | 7S | 8S | RHS
----+-----+----+----+----+----+----+----+----+----+-----
  1 |   5 |  2 | -1 |  1 |  0 |  1 |  0 |  0 |  0 |  30
  2 |   6 |  3 |  1 | -1 |  0 |  0 |  1 |  0 |  0 |  26
  3 |   7 |  0 |  1 |  1 |  0 |  0 |  0 |  1 |  0 |  13
  4 |   8 |  0 |  0 |  0 |  1 |  0 |  0 |  0 |  1 |  20
----+-----+----+----+----+----+----+----+----+----+-----
 Zj |     | -4 |  2 | -1 | -5 |  0 |  0 |  0 |  0 |   0

   Zj positive does exist.
=============================================================================================================
=

   Iteration No.: 1
=============================================================================================================
=
   k= 2, r= 3

  i | BVS |  1 | 2 |  3 |  4 | 5S | 6S | 7S | 8S | RHS
----+-----+----+---+----+----+----+----+----+----+-----
  1 |   5 |  2 | 0 |  2 |  0 |  1 |  0 |  1 |  0 |  43
  2 |   6 |  3 | 0 | -2 |  0 |  0 |  1 | -1 |  0 |  13
  3 |   2 |  0 | 1 |  1 |  0 |  0 |  0 |  1 |  0 |  13
  4 |   8 |  0 | 0 |  0 |  1 |  0 |  0 |  0 |  1 |  20
----+-----+----+---+----+----+----+----+----+----+-----
 Zj |     | -4 | 0 | -3 | -5 |  0 |  0 | -2 |  0 | -26

   Zj positive does NOT exist.
=============================================================================================================
=

      R E S U L T   T E S T I N G
```

```
================================================================================================
=

   constraint no.   1 TRUE : -13 <= 30
   constraint no.   2 TRUE : 13 <= 26
   constraint no.   3 TRUE : 13 <= 13
   constraint no.   4 TRUE : 0 <= 20

   Test result: OK

================================================================================================
=

      S U M M A R Y
================================================================================================
=

   Status: OPTIMAL

   Degeneracy: NO

   Test to constraints: OK

   MIN: -26

   BVS: [5, 6, 2, 8 ]

   x(1) = 0
   x(2) = 13
   x(3) = 0
   x(4) = 0

   Number of iterations: 1
   Number of additions/subtractions: 117
   Number of multiplications/divisions: 132
   Number of loops: 155
   Number of decisions: 118



================================================================================================
=
```

PRILOGA 7: Seznam v delu uporabljenih kratic

1. LP − linearno programiranje, problemi linearnega programiranja;
2. BVS − množica baznih vektorjev;
3. RHS − vrednosti desnih strani neenačb;
4. CR − količnik stolpca
5. RR − količnik vrstice
6. DLL − dinamična programska knjižnica