Универзитет у Београду
Математички факултет

Драган Д. Ђурђевић

Поређење егзактних и хеуристичких метода за решавање неких оптимизационих проблема

Мастер рад

Београд, 2014.

Ментор:

др Филип Марић, доцент, Универзитет у Београду, Математички факултет

Чланови комисије:

др Предраг Јаничић, ванреди професор, Универзитет у Београду, Математички факултет

др Мирослав Марић, доцент, Универзитет у Београду, Математички факултет

Датум одбране: ____.10.2014.

Наслов:

Поређење егзактних и хеуристичких метода за решавање неких оптимизационих проблема

Резиме:

Питање оптималног управљања и оптималног коришћења расположивих ресурса од изузетног је значаја у савременом свету и у различитим областима људског деловања. Појавом све већег броја оптимизационих проблема у практичним применама и знатним повећањем димензија проблема појавило се питање развоја што ефикаснијих метода за њихово решавање. Ово је посебно значајно за проблеме целобројног програмирања. Развијен је велики број различитих метода које се могу поделити на егзактне и хеуристичке. Због мноштва доступних метода занимљиво је питање испитати ефикасност различитих метода примењених за решавање истог проблема, јер постоји могућност да су одређене методе погодније за решавање неког проблема. У овом раду се посвећује пажња коришћењу егзактних метода које нису толико уобичајене као методе за решавање оптимизационих проблема и то SAT, SMT и CSP решавача, и поређењу њихове ефикасности у односу на стандардне егзактне методе оптимизације садржане у CPLEX пакету, као и у односу на метахеуристичке методе генетски алгоритам и локалну претрагу, као и њихову комбинацију. Циљ рада је да испита да ли је примена SAT, SMT и CSP решавача за решавање неких оптимизационих проблема оправдана. Методе су тестиране на проблему двонивоске локације постројења са неограниченим капацитетима. Резултати показују да је CPLEX најпогоднији за решавање разматраног проблема, док је само неке од наведених алтернативних егзактних метода могуће користити. Коришћење хеуристичких метода оправдано је у случају инстанци великих димензија које CPLEX-у могу представљти проблем.

Кључне речи:

оптимизација, SAT, SMT, CSP, CPLEX, генетски алгоритам, локална претрага

Садржај

1. Увод

Питање оптималног управљања и оптималног коришћења расположивих ресурса од изузетног је у савременом свету и многим областима људског деловања. Оптималност подразумева да су сва доступна средства искоришћена на такав начин да су у највећој могућој мери допринела остварењу одређеног циља. Оптимизациони проблем је проблем у ком се захтева максимално или минимално постизање одређеног циља користећи доступна средства.

У овим терминима могуће је дефинисати најразличитије проблеме од којих многи имају значајне практичне примене. Као пример могуће је навести: проблем ранца (разломљени и целобројни), проблем клика, проблем распореда послова на процесоре, друге проблеме распоређивања, различите локацијске проблеме и многе многе друге проблеме који се јављају у економији [14], пољопривреди [16], индустрији [15] и другим областима.

Проблем математичке оптимизације, а присутан је и термин проблем математичког програмирања, је проблем налажења минимума функције циља при чему се захтева важење одређеног скупа ограничења тј. скупа неједначина. Ограничења формирају такозвани простор допустивих решења S (некад се назива и простор претраге). Тада се оптимизациони проблем може записати у облику:

$$\min \{ f(x) \mid x \in S \},$$
$$x = (x_1, \ldots, x_n)$$

Уколико за неко $x_0 \in S$ важи $\forall x \in S, f(x_0) \leq f(x)$ онда је то x_0 глобални оптимум.

У зависности од особина функције циља и ограничења разликује се неколико врста оптимизационих проблема од којих помињемо само неке. Тако се проблемом линеарног програмирања назива проблем математичке оптимизације код ког су и функција циља и ограничења задати као линеарне функције. Проблеми

целобројног програмирања су проблеми код којих се захтева да је простор допустивих решења потпростор простора над скупом целих бројева. Комбинаторна оптимизација се бави проблемима код којих је скуп допустивих решења коначан. У овом раду ће пажња бити посвећена проблемима целобројног и мешовитог целобројног програмирања (то су проблеми који могу имати и дискретне и континуалне променљиве).

Методе за решавање оптимизационих проблема се могу поделити на две групе. У прву групу спадају егзактне методе. То су методе које за пронађено решење гарантују да је баш то решење глобални оптимум на скупу допустивих решења. Међутим, један број оптимизационих проблема, а нарочито проблеми целобројног програмирања, припада класи NP-тешких проблема што значи да за њихово решавање још увек није пронађен алгоритам који се извршава у полиномијалном времену.

Када се говори о проблемима у класи NP мисли се на проблеме одлучивања. Проблеми одлучивања су проблеми чије је решење одговор да/не. Проблем одлучивања је у класи NP ако је могуће у полиномијалном времену прихватити решење за које је одговор да. У класи NP постоји посебна класа проблема – NP комплетни проблеми. Проблем је NP комплетан ако припада класи NP и сви проблеми класе NP могу се свести на тај проблем. За NP комплетне проблеме још увек није пронађен алгоритам који их решава у полиномијалном времену.

Оптимизациони проблеми нису проблеми одлучивања али се могу свести на проблеме одлучивања. Оптимизациони проблем се може свести на проблем одлучивања увођењем питања да ли је вредност функције циља мања од неке вредности K. Проблем H је NP-тежак ако и само ако постоји NP-комплетан проблем H' који је полиномијално сводљив на проблем H. Пошто за NP-комплетне проблеме није пронађен алгоритам који их решава у полиномијалном времену то ни за NP-тешке проблеме није пронађен ефикасан егзактан алгоритам. Због тога, уопштено посматрано, егзактне методе нису погодне за решавање NP-тешких

оптимизационих проблема великих димензија, нпр. неких проблема комбинаторне оптимизације.

У другу групу спадају различите хеуристичке и метахеуристичке методе. Хеуристичке методе су методе које су прилагођене решавању конкретног оптимизационог проблема или групе проблема. Метахеуристичке методе се могу прилагодити за решавање великог броја оптимизационих проблема и могуће их је сматрати скупом упутстава за формирање хеуристичких метода које су прилагођене конкретном проблему. Хеуристичке и метахеуристичке методе су обично много ефикасније у погледу времена извршавања од егзактних метода. Међутим, ове методе не дају никакву гаранцију да је добијено решење глобални оптимум на скупу допустивих решења. Иако неретко дају решења која су блиска оптималном, па чак и достижу оптимална решења, недостатак гаранције достизања глобалног оптимума је упадљив недостатак.

2. Коришћене методе и појмови

Интересантно је извршити поређење различитих метода за решавање оптимизационих проблема јер одређене методе могу бити погодније за решавање неких проблема. Егзактне методе могу бити добро решење али вероватно не могу решити проблеме великих димензија у разумном времену. Ипак, и међу егзактним методама може бити метода које су из неког разлога ефикасније за решавање одређених проблема. Слично важи и за метахеуристичке методе. Неком проблему може више одговарати одређена метахеуристика а нека друга може имати лошије резултате.

У овом раду се посвећује пажња коришћењу егзактних метода које нису толико уобичајене као методе за решавање оптимизационих проблема и то SAT, SMT и CSP решавача, и поређењу њихове ефикасности у односу на стандардне егзактне методе оптимизације садржане у CPLEX пакету, као и у односу на метахеуристичке методе генетски алгоритам и локалну претрагу, као и њихову комбинацију.

2.1. SAT и SAT-решавачи

Исказна логика је изузетно значајна област математике. Иако је језик исказне логике прилично сиромашан ипак се може користити за моделовање великог броја проблема. У овом раду ћемо видети како је могуће оптимизациони проблем моделовати користећи исказну логику и решавати га коришћењем алгоритама за испитивање задовољивости исказних формула. Пре тога упознаћемо се са основним појмовима исказне логике и испитивања задовољивости исказних формула.

Неки од познатијих SAT решавача су: minisat, SATzila, Lingeling.

2.1.1. Језик исказне логике

На језику исказне логике можемо записати различите проблеме. Основни појам исказне логике је исказ. Исказ представља тврђење за коју се може утврдити да ли је тачно или нетачно. Искази могу имати сложену структуру коју сачињавају исказне променљиве повезане исказним везницима. Исказне променљиве представљају атомичке исказе у контексту исказне логике тј. исказе чија се евентуално сложена структура даље не мора разматрати. Разликујемо синтаксу и семантику исказне логике.

Синтакса дефинише на који начин се записују исправне језичке конструкције. Језичке конструкције исказне логике су исказне формуле. Исказне формуле су на најнижем нивоу састављене од исказних слова повезаних исказним везницима.

Формално, језик исказне логике се дефинише над скупом симбола који сачињава коначан скуп исказних слова P, скуп исказних константи и скуп исказних везника.

$$\Sigma = \{p_1, \ldots, p_n\} \cup \{0,1\} \cup \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$$

Језик исказне логике представља скуп исказних формула. Исказна формула се дефинише рекурзивно на следећи начин:

- 0, $\top$, $p \in P$ је исказна формула
- ако су A и B исказне формула онда су исказне формуле и $\neg(A), (A) \wedge (B), (A) \vee (B), (A) \Rightarrow (B), (A) \Leftrightarrow (B)$

Исказна слова и исказне константе називају се атомичке формуле. Атомичке формуле и негације атомичких формула се називају литерали. Дисјункција литерала назива се клауза. Ако је p исказно слово кажемо да је формула p литерал позитивног поларитета а формула $\neg p$ литерал негативног поларитета. Супротан литерал литералу p је литерал $\neg p$ и обратно. Често се за литерал супротног поларитета литералу l користи ознака $\bar{l}$.

Супституција формуле C формулом D у формули F, у ознаци $F[C \to D]$, се

дефинише на следећи начин:

- ако је $F = C$ онда је $F[C \to D] = D$
- ако је F исказно слово или исказна константа онда је $F[C \to D] = F$
- ако је $F = \neg(B)$ онда је $F[C \to D] = \neg(B[C \to D])$
- ако је $F = A \wedge B$, $F = A \vee B$, $F = A \Rightarrow B$ или $F = A \Leftrightarrow B$ онда је
  $F[C \to D] = (A[C \to D]) \wedge (B[C \to D])$,
  $F[C \to D] = (A[C \to D]) \vee (B[C \to D])$,
  $F[C \to D] = (A[C \to D]) \Rightarrow (B[C \to D])$,односно
  $F[C \to D] = (A[C \to D]) \Leftrightarrow (B[C \to D])$

Семантика исказне логике подразумева давање значења исказним формулама.

Валуација представља функцију која пресликава скуп исказних слова P у скуп $\{0,1\}$. Интерпретација представља функцију која пресликава скуп исказних формула над скупом исказних слова P при валуацији v у скуп $\{0,1\}$ на следећи начин:

- $I_v(p) = v(p), p \in P$
- $I_v(\bot) = 0$ и $I_v(\top) = 1$
- $I_v(\neg A) = 0$ *ако је* $I_v(A) = 1$, *и* $I_v(\neg A) = 1$ *ако је* $I_v(A) = 0$
- $I_v(A \wedge B) = 1$ *ако је* $I_v(A) = 1$ *и* $I_v(B) = 1$, *иначе* $I_v(A \wedge B) = 0$
- $I_v(A \vee B) = 0$ *ако је* $I_v(A) = 0$ *и* $I_v(B) = 0$, *иначе* $I_v(A \vee B) = 1$
- $I_v(A \Rightarrow B) = 0$ *ако је* $I_v(A) = 1$ *и* $I_v(B) = 0$, *иначе* $I_v(A \Rightarrow B) = 1$
- $I_v(A \Leftrightarrow B) = 1$ *ако је* $I_v(A) = 1$ *и* $I_v(B) = 1$ *или* $I_v(A) = 0$ *и* $I_v(B) = 0$
  иначе $I_v(A \Leftrightarrow B) = 0$

Кажемо да је формула F тачна у валуацији v ако важи $I_v(F) = 1$, у супротном кажемо да је формул нетачна у валуацији. Кажемо да је формула задовољива ако постоји валуација таква да је формула тачна. У случају да таква валуација постоји кажемо да је та валуација модел формуле. Кажемо да је формула порецива у

сличају да постоји валуација у којој је формула нетачна. Кажемо да је формула таутологија (ваљана) ако је тачна за сваку валуацију. Кажемо да је формула контрадикција ако је нетачна за сваку валуацију.

За две формуле $F_1 u F_2$ кажемо да су логички еквивалентне, у ознаци $F_1 \equiv F_2$, ако и само ако за сваку валуацију v важи $I_v(F_1) = I_v(F_2)$. За две формуле $F_1 u F_2$ кажемо да су еквизадовољиве ако и само ако када постоји модел за F_1 тада постоји модел за F_2.

Кажемо да је валуација v модел скупа исказних формула $\Gamma = \{A_1, \ldots, A_n\}$ ако и само ако је v модел сваке формуле из скупа Γ. Кажемо да је формула F логичка последица скупа формула Γ ако и само ако је сваки модел за скуп формула Γ уједно и модел за F. Важи: формула F је логичка последица скупа формула Γ ако и само ако је $A_1 \wedge \ldots \wedge A_n \Rightarrow F$ таутологија.

За поједине примене погодно је исказне формуле представити у специјалним облицима који се називају нормалне форме. За испитивање задовољивости исказних формула модерни решавачи користе формуле у конјунктивној нормалној форми (КНФ). Формула је у конјуктивној нормалној форми ако је конјункција дисјункција тј. ако је облика:

$$A_1 \wedge \ldots \wedge A_n, \textit{ где је } A_i \textit{ клауза.}$$

Често је погодно формуле представљене у КНФ посматрати као скупове клауза а клаузе као скупове литерала.

Значајно је што је од сваке исказне формуле F могуће добити еквизадовољиву формулу F' у полиномијалом времену коришћењем супституције подформула формуле F њима логички еквивалентним формулама како би формула биле поједностављена и применом Цајтинове трансформације. За детаљан опис поступка трансформације произвољне формуле у КНФ може се погледати нпр. [1] и [11].

2.1.2 Испитивање задовољивости исказне формуле

Испитивање задовољивости исказне формуле једно је од најважнијих питања теоријског рачунарства. То је први проблем за који је показано да је NP-комплетан. То је и централни проблем исказне логике.

Проблем испитивања да ли је исказна формула таутологија своди се на проблем испитивања задовољивости. Наиме, ако постоји валуација у којој формула није тачна формула није таутологија. У тој валуацији негација формуле биће тачна тј. негација формуле биће задовољива. Дакле, испитивање да ли је формула таутологија своди се на испитивање да ли је негација формуле задовољива. Слично се и испитивање да ли је формула контрадикција и да ли је формула порецива своде на испитивање да ли је формула задовољива.

SAT проблем (енг. Satisfiability problem) се дефинише као испитивање задовољивости исказне формуле у КНФ.

Данас постоји неколико SAT-решавача. Већина је заснована на DPLL процедури али постоје и стохастички SAT-решавачи.

Испитивање задовољивости исказне формуле је NP-комплетан проблем. То значи да је сложеност до сада познатог најефикаснијег алгоритма за решавање овог проблема експоненцијална, у овом случају $O(2^n)$ где је n број литерала. Ипак, најгори случај при испитивању задовољивости је случај када морају бити проверене све валуације којих има експоенцијално, али овај случај се не дешава често. У случају да формула јесте задовољива довољно је пронаћи макар једну валуацију која је модел формуле што може захтевати знатно мање времена.

При испитивању задовољивости неопходно је вршити додељивање вредности исказним променљивим. Да би се унапредила ефикасност потребно је увести и одређене елементе закључивања пошто постављање вредности неких

променљивих може одредити вредности клауза или чак целе формуле. Увођењем тих елемената формирана је DPLL (Davis-Patman-Logemann-Loveland) процедура. Елементи закључивања које уводи DPLL процедура су: корак таутологије (енг. tautology), јединична пропагација (енг. unit propagate) и пропагација чистих литерала (енг. pure literal).

Када у клаузи постоји литерал чија је вредност постављена на нетачно тај литерал се може уклонити из клаузе јер вредност клаузе неће зависити од тог литерала. Корак таутологије подразумева да се клаузе у којима је макар један литерал има постављену вредност тачно уклањају из формуле јер оне тада имају вредност тачно без обзира на вредности осталих литерала. Оба ова корака следе на основу особина дисјункције. Ако након корака уклањања литерала из клаузе добијемо празну клаузу то значи да су сви литерали те клаузе били постављени на вредност нетачно. У том случају је и цела клауза нетачна, па је и формула у КНФ нетачна тј. формула у тој валуацији није задовољива. Ако након корака таутологије формула постане празна то значи да су све клаузе формуле уклоњене применом корака таутологије тј. да су све клаузе тачне па је тада и формула у КНФ тачна тј. добили смо валуацију која задовољава полазну формулу.

Корак јединичне пропагације се односи на јединичне клузе. Клауза је јединична ако садржи само један литерал. У том случају је неопходно да тај литерал буде постављен на вредност тачно да би формула била задовољива.

Чист литерал је литерал који се у свим клаузама формуле налази само у једном поларитету. Пошто се налази у само једном поларитету погодно је, мада не мора бити неоходно, поставити његову вредност на тачно јер на тај начин добијамо један број тачних клауза.

Ако није могуће применити ни један од ових корака на крају се примењује правило раздвајања (енг. split). Одабира се један литерал и најпре се њему додељује вредност тачно. Ако је формула након те доделе задовољива враћа се

одговор да је формула задовољива. У супротном, поставља се вредност истог литерала на нетачно и испитује се задовољивост тако добијене формуле. Овим правилом је омогућено прегледање свих могућих валуација тј. валуација које нису елиминисане претходним корацима.

Псеудокод основне DPLL процедуре [1]:

```
bool DPLL(Formula f)
{
     if(f je prazna)
          return true;
     svako ¬0 u f zameniti sa 1 i svako ¬1 u f zameniti sa
 0
     ukloniti sve literale 0
     if(u f postoji prazna klauza)
          return false;
     if(postoji klauza k sa literalom l)
          return DPLL(f \ k);
     if(postoji jedinicna klauza sa literalom l u f)
          return DPLL(f[ l→1 ]);
     if(postoji cist literal l u f)
          return DPLL(f[ l→1 ]);
     odabrati literal l;
     if(DPLL(f[ l→1 ]))
          return true;
     else
          return DPLL(f[ l→0 ]);
}
```

DPLL процедура се у оваквом облику данас ретко примењује. Мада већина решавача почива на овим основама они уводе значајне новине. Једна новина је

изостављање правила пропагације чистих литерала. Испоставља се да је често провера да ли је неки литерал чист временски захтевна и да корисност правила не оправдава његову цену.

Недостатак је и што је оригинална процедура рекурзивна и подразумева преношење формуле као параметра функције. Дубина рекурзије може бити значајна а и формула у савременим применама може имати велики број клауза и литерала тако да овакав приступ може бити неефикасан. Овај проблем се решава прослеђивањем валуације која се изграђује као параметра функције, а затим се и то елиминише симулирањем рекурзије коришћењем стека који представља парцијалну валуацију коју решавач инкрементално изграђује.

Избор литерала у кораку раздвајања може бити изузетно значајан. Добрим избором литерала може се знатно убрзати долазак до решења. За избор литерала се код савремених решавача користе различите хеуристике.

Увођењем савремених структура података може се знатно убрзати провера да ли је клауза постала јединична. За то се користи струкутура података два посматрана литерала.

И коначно, може се увести учење. Неке доделе вредности литералима могу бити проблематичне тј. могу изазвати конфликте без обзира на то какве доделе вредности другим литерала су им претходиле. Те проблематичне доделе могу бити повезане у клаузе и додате формули ради раног откривања проблематичних додела чиме се побољшава ефикасност. Учење се врши применом исказне резолуције. Ипак, да би се спречио неограничени раст формуле учење се држи под контролом применом механизма заборављања тако што се неке научене клаузе након одређеног времена бришу. Некада се након неког времена и након одређеног броја научених клауза претрага поново покреће, али сада уз присуство научених клауза са претпоставком да ће научене клаузе допринети бржем доласку до решења.

2.2 SMT и SMT-решавачи

Логика првог реда, позната и као предикатска логика, представља проширење исказне логике у односу на коју доноси још веће изражајне могућности. Ипак, цена која је морала бити плаћена је да је централни проблем логике првог реда, испитивање ваљаности произвољне формуле логике првог реда, је неодлучив. Проблем одлучивања је неодлучив ако није могуће конструисати алгоритам такав да за сваку инстанцу проблема алгоритам враћа одговор. За логику првог реда важи полуодлучивост тј. за сваку ваљану формулу могуће је доказати њену ваљаност.

Ипак, често није битно питање да ли је формула ваљана већ да ли је формула ваљана у некој теорији тј. фрагменту логике првог реда. Постоје фрагменти логике првог реда који су одлучиви као што су теорија неинтерпретираних функцијских симбола са једнакошћу и, што је за тему овог рада значајније, линеарна аритметика.

Овим питање бави се област задовољивости у односу на теорију (SMT - енг. Satisfiability Modulo Theories). Подразумева комбиновање савремених SAT-решавача и решавача специфичних за одређену теорију. Неки од познатијих су: yices, Z3, CVC3, MathSat.

2.2.1 Логика првог реда

Већу изражајну моћ у односу на исказну логику логика првог реда дугује богатијем језику. У исказној логици су искази посматрани као елементарне тврдње. У логици првог реда они могу поседовати и одређену унутрашњу структуру и називају се предикати. Опис сложености унутрашње структуре

омогућен је употребом функцијских симбола. Увођењем квантификатора омогућено је записивање тврђења која нису могла бити записана у исказној логици.

Разликујемо синтаксу и семантику логике првог реда. Синтакса логике првог реда се изграђује над скупом променљивих *V*, скупом логичких симбола $\{\bot, \top, \exists, \forall, \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$, и сигнатуром (језиком) коју чине: скуп функцијских симбола *F*, скуп предикатских (релацијских) симбола *P*, и функција *ar* која сваком функцијском симболу из *F* и сваком предикатском симболу из *P* придружује арност.

Термови логике првог реда изграђују се на следећи начин:

- променљива $p \in P$ је терм
- константа (функцијски симбол арности 0) је терм
- ако су $t_1, \ldots, t_n$ термови и *f* функцијски симбол арности *n* онда је и $f(t_1, \ldots, t_n)$ терм

Атомичке формуле логике првог реда граде се на следећи начин:

- логичке константе су атомичке формуле
- ако су $t_1, \ldots, t_n$ термови и *p* предикатски симбол арности *n* онда је атомичка формула и $p(t_1, \ldots, t_n)$

Формуле логике првог реда добијају се на следећи начин:

- атомичке формуле су формуле
- ако су *A* и *B* формуле онда су формуле и $\neg(A), (A) \wedge (B), (A) \vee (B), (A) \Rightarrow (B), (A) \Leftrightarrow (B)$
- ако је A формула и *x* променљива онда су формуле и $(\forall x)(A), (\exists x)(A)$

Кажемо да је променљива везана ако је под дејством неког квантификатора, у супротном кажемо да је променљива слободна. Формуле које немају слободне

променљиве називају се реченице. Формуле које не садрже променљиве називају се базне формуле.

За одређивање истинитосне вредности формуле у логици првог реда је поред познавања вредности промемљивих неопходно знати и на који начин се интерпретирају функцијски и предикатски симболи, као и које вредности могу узети променљиве.

Нека је дата сигнатура *L*. *L*-структуру **D** чине:

- непразан скуп *D* који се назива домен
- за сваку константу *c* њена интерпретација $c_{\boldsymbol{D}} \in D$
- за сваки функцијски симбол *f* арности *n* његова интерпретација $f_{\boldsymbol{D}}: D^n \rightarrow D$
- за сваки предикатски симбол *p* арности *n* његова интерпретација $p_{\boldsymbol{D}} \subseteq D^n$

Валуација за структуру **D** је функција којом се променљивим додељује нека вредност из домена *D*.

Вредност терма у структури **D** и при валуацији *v* дефинише се на следећи начин:

- ако је терм *t* променљива *p* онда је његова вредност $\boldsymbol{D}_v(t) = v(p)$
- ако је терм *t* константа *c* онда је његова вредност $\boldsymbol{D}_v(t) = c_{\boldsymbol{D}}$
- ако је терм *t* облика $f(t_1, \ldots, t_n)$ онда је његова вредност $\boldsymbol{D}_v(t) = f_D(\boldsymbol{D}_v(t_1), \ldots, \boldsymbol{D}_v(t_n))$

Тачност формуле *F* у структури **D** и при валуацији *v* се дефинише на следећи начин:

- ако је облика $p(t_1, \ldots, t_n)$ онда је тачна ако важи $p_{\boldsymbol{D}}(\boldsymbol{D}_v(t_1), \ldots, \boldsymbol{D}_v(t_n))$
- ако је 1 тачна је при свакој интерпретацији и у свакој валуацији, а ако је 0 нетачна је при свакој интерпретацији и у свакој валуацији

- ако је облика $\neg A$ тачна је ако је формула A нетачна у структури **D** и при валуацији v, инаце је нетачна
- ако је облика $A \wedge B$ тачна је ако и само ако је A тачна у структури **D** и при валуацији v и B је тачна у структури **D** и при валуацији v, иначе је нетачна
- ако је облика $A \vee B$ нетачна је ако и само ако је A нетачна у структури **D** и при валуацији v и B је нетачна у структури **D** и при валуацији v, у супротном је тачна
- ако је облика $A \Rightarrow B$ нетачна је ако и само ако је A тачна у структури **D** и при валуацији v и B је нетачна у структури **D** и при валуацији v, у супротном је тачна
- ако је облика $A \Leftrightarrow B$ тачна је ако и само ако је A тачна у структури **D** и при валуацији v и B је тачна у структури **D** и при валуацији v или ако је A нетачна у структури **D** и при валуацији v и B је нетачна у структури **D** и при валуацији v, у супротном је нетачна
- ако је облика $(\exists x)(F)$ тачна је ако и само ако постоји валуација v' добијена од v изменом вредности променљиве x таква да је F тачна у структури **D** при валуацији v'
- ако је облика $(\forall x)(F)$ тачна је ако и само ако је F тачна у структури **D** при свакој валуација v' добијеној од v изменом вредности променљиве x

Формула логике првог реда је ваљана ако је тачна при свакој интерпретацији у свакој валуацији.

Ваљаност се испитује методом побијања која је обично заснована на методу резолуције. При доказивању побијањем доказује се да је негација полазне формуле незадовољива па полазна формула мора бити ваљана. Формула се преводи у пренекс нормалну форму, врши се сколемизација а затим и превођење у клаузалну нормалну форму. Након тога се из скупа добијених клауза методом резолуције за логику првог реда, уз коришћење унификације, покушава извести празна клауза. Према теореми о комплетности метода резолуције за логику првог реда ако је

формула у клаузалној нормалној форми незадовољива методом резолуције је могуће извести празну клаузу. Ако је формула задовољива метод резолуције може да се не заустави па зато није метод одлучивања за логику првог реда. Заправо, такав метод ни не постоји.

2.2.2 Испитивање задовољивости формуле логике првог реда у односу на теорију

Као што се види, ваљаност је изузетно јако тврђење. Ваљана формула је тачна у свакој интерпретацији и свакој валуацији, што значи да је тачна у било којој теорији која може бити формализована у терминима логике првог реда. У неким применама тако јака тврђења нису неопходна. Довољно је испитати да ли је формула ваљана или задовољива у одређеној теорији или фрагменту логике првог реда.

Када се поставе одређена ограничења над обликом формуле, сигнатуром, интерпретацијом нелогичких симбола и слична добијају се фрагменти логике првог реда. Примери могу бити линеарна аритметика (реална, целобројна), теорија једнакости са неитерпретираним функцијским симболима, теорија низова или теорија листи. Такви фрагменти могу бити одлучиви. Ако је формула тачна у неком фрагменту тј. теорији кажемо да је теорема те теорије.

И раније су постојали методи који су испитивали да ли је нека формула одређеног фрагмента задовољива. Заправо, то су класични методи за решавање проблема у различитим теоријама. Развојем SAT-решавача јавила се идеја о комбиновању метода одлучивања за конкретну теорију и модерних SAT-решавача. Тако је настало SMT поље изучавања. Уобичајено се комбинују SAT-решавачи и методе одлучивања који су у стању да реше конјункције литерала теорије T којој су намењени такозвани T-решавачи [4][5]. Ово је такозвани DPLL(T) приступ.

Комбиновање SAT-решавача и T-решавача се врши тако што се најпре атоми формуле теорије посматрају као искази. Треба напоменути да је уобичајен захтев да формула буде формула без квантификатора. На тај начин се добија исказна формула која се предаје SAT-решавачу. Ако SAT-решавач утврди да је формула исказно незадовољива онда она мора бити и незадовољива и у оквиру теорије. Ако је формула исказно задовољива могуће је добити њен исказни модел. У том моделу се врши замена исказних слова атомима оригиналне формуле. На тај начин се добија конјункција литерала теорије T тј. нова, помоћна формула. Та формула се онда предаје T-решавачу. Ако T-решавач утврди да је та формула задовољива, онда је и оригинална формула задовољива и могуће је наћи њен модел. У супротном се помоћна формула негира и додаје се полазној формули. Та измењена полазна формула се предаје SAT-решавачу и поступак се понавља.

Овом основном приступу могуће је додати нека унапређења [4]. Како модерни SAT-решавачи инкрементално изграђују парцијалну валуацију не мора се чекати да модел буде потпуно изграђен. Могуће је при додавању литерала у валуацију извршити позив T-решавача. У случају да T-решавач открије да је парцијална валуација незадовољива у теорији могуће је одмах урадити повратни скок и избећи непотребно проверавање великог броја валуација. Ово је такозвани приступ инкременталног T-решавача (енг. incremental T-solver).

Друго могуће побољшање је да при откривању незадовољивости у теорији није неопходно започети SAT претрагу из почетка већ се вратити у неко стање које је претходно било задовољиво у односу на теорију. Ово је такозвани активни SAT-решавач (енг. on-line SAT-solver).

Још једно могуће побољшање укључује могућност пропагације теорије (енг. theory propagation). Наиме, при одређеној парцијалној валуацији T-решавач може бити позван како би били утврђени неки литерали који су логичка последица до тада изграђеног модела. Те информације могу бити прослеђене SAT-решавачу како би претрага била усмерена у корисном правцу.

Као Т-решавачи за линеарну аритметику, што је за овај рад од највећег значаја, користе се методи базирани на симплексу или Фуријe-Моцкиновој елиминацији [6]. Нпр. yices користи метод базиран на симплексу а за проблеме са целобројним ограничењима метод гранања и одсецања, који је комбинација метода гранања и ограничавања и метода одсецања. Како исти основни метод користи и CPLEX пакет о методу гранања и ограничавања ће више речи бити у 2.4.1.

2.3 CSP и CSP решавачи

Неке врсте проблема се врло лако описују формулисањем скупа ограничења које решење проблема треба да задовољава. У извесном смислу и SAT-проблем се може посматрати као проблем задовољавања ограничења ако се КНФ посматра као скуп ограничења, клауза као ограничење а модел формуле као решење које задовољава задати скуп ограничења. Слично, као проблеми задовољавања ограничења се могу посматрати и оптимизациони проблеми.

Неки од CSP решавача су: minion, Gecode, G12, Opturion CPX, Choco.

2.3.1 Проблем задовољавања ограничења

Проблем задовољавања ограничења се дефинише тако што је дат скуп променљивих $x_1, \ldots, x_n$ које узимају вредности из одговарајућих домена $D_1, \ldots, D_n, D_i \neq \emptyset, i = 1, \ldots, n$ и над њима су дефинисана ограничења $C_1, \ldots, C_m$ [8]. Стање је одређено вредностима додељеним неким или свим променљивим. Стање које не нарушава ни једно од задатих ограничења назива се конзистентно стање. Решење је стање у ком су свим променљивим додељене вредности и то стање задовољава сва ограничења, или доказ да такво решење не постоји.

У зависности од природе домена разликујемо континуални проблем задовољавања ограничења, чији је један представник проблем линеарног програмирања, и дискретни проблем задовољавања ограничења. Дискретни проблем може бити коначан, ако су домени коначни, или бесконачан. Ако су домени свих променљивих такви да имају само две вредности тачно и нетачно ради се о Буловском проблему задовољавања ограничења.

Ограничења могу бити унарна, ако укључују само једну променљиву, бинарна, ако укључују више променљивих, или ограничења вишег реда. Глобална ограничења су посебна врста ограничења вишег реда која укључују све променљиве. Проблем задовољавања ограничења који има само бинарна ограничења назива се бинарни проблем. Бинарни проблем задовољавања ограничења може бити представљен графом у ком су чворови променљиве а гране представљају ограничења у којима се променљиве јављају. Остали проблеми могу бити представљени хиперграфом или бипартитним графом у ком један скуп чворова представљају променљиве а други ограничења. Гране тог графа повезују променљиву и ограничења у којима учествује. Ово је такозвани граф конзистентности.

2.3.1 Решавање проблема задовољавања ограничења

Проблем задовољавања ограничења може бити решен на неколико начина. Различити CSP-решавачи имају различите приступе који укључују свођење на SAT, свођење на SMT, свођење на гранање и ограничавање и свођење на симплекс или коришћење метода који узимају у обзир одлике CSP проблема као што је метод гранања и пропагације.

Детаљније ће бити описан метод гранања и пропагације специфичан са решавање проблема задовољавања ограничења.

Метод гранања и пропагације могуће је применити на проблеме задовољавања ограничења код којих све променљиве имају коначне домене. Ово је и најчешћи тип проблема ограничавања и представља уопштење неких NP-комплетних проблема, на пример SAT, па није познат општи алгоритам који ову врсту проблема решава у полиномијалном времену.

Метод гранања и пропагације полази од празног стања и формира стабло претраге у чијем корену је празно стање. За сваку променљиву памти се скуп дозвољених вредности, иницијално је то цео домен. У сваком чвору стабла претраге бира се једна променљива којој вредност није додељена и врши се додела вредности тој променљивој из скупа дозвољених вредности. Додељена вредност се уклања из скупа. Ако је стање неконзистентно претрага се враћа. Ако је стање потпуно тј. додељена је вредност свим променљивим и при том није нарушено ни једно ограничење проблем је решен.

Сложеност оваквог приступа је $O(d^n)$ где је број променљивих означен са n а $d = max\{|D_i|, ..., |D_n|\}$.

Најзначајнији део овог метода представљају пропагације тј. приликом доделе вредности некој променљивој користећи ограничења могуће је извршити смањење скупова дозвољених вредности других променљивих. Неке пропагације је могуће извршити чак и пре почетка претраге.

Једна од метода пропагације је провера унапред (енг. forward checking). Након доделе вредности некој променљивој x проверавају се све променљиве које се јављају у истим ограничењима као променљива x а којима вредности нису додељене. Вредности из скупа дозвољених решења тих променљивих се бришу уколико нису конзистентне са вредношћу која је додељена променљивој x. Ако се при том за неку променљиву добије празан скуп дозвољених решења врши се враћање уназад. Овако је могуће знатно раније открити неконзистентности.

Друга група метода пропагације је метод пропагације ограничења и подразумева пропагацију последица ограничења над једном променљивом на друге променљиве [8]. Једна од оваквих метода је конзистентност ивице (енг. arc consistency). Назив се односи на усмерену ивицу графа конзистентности (за бинарни проблем или ограничења у којим су остале само две променљиве којима није додељена вредност). Ивица која повезује две променљиве *x* и *y* је конзистентна ако за сваку вредност из скупа дозвољених вредности за *x* постоји вредност за *y* која не нарушава конзистентност. Ако ивица није конзистентна може се учинити конзистентном избацивањем вредности из скупа дозвољених вредности за променљиву *x* за које не постоји конзистентна вредност променљиве *y*. Приликом сваког избацивања вредности за *x* потребно је поново извршити проверу конзистентности за све остале променљиве јер свако уклањање вредности из скупа дозвољених вредности за *x* може изазвати нову неконзистентност.

Могуће је вршити и проверу k-конзистентности. Проблем је k-конзистентностан ако за сваки *k-1* подскуп променљивих којима су додељене вредности и који је конзистентан свакој преосталој променљиви може бити додељена конзистентна вредност. Проблем је јако k-конзистентностан ако је k-конзистентностан, k-1 -конзистентностан, k-2 -конзистентностан, …, 1-конзистентан. Иако провера k-конзистентности може смањити број гранања одузима превише времена.

Ово је основни метод и њему се могу додати одређена унапређења. Једно од унапређења односи се на хеуристику којом се бира променљива којој се додељује вредност. Једна од метода, такозвана метода најмање преосталих вредности (енг. minimum remaining values), је да се бира променљива чији је скуп дозвољених вредности најмањи. На овај начин се покушавају направити велика одсецања простора претраге.

При избору променљиве може се десити да су неким променљивим скупови дозвољених вредности једнаке величине. У том случају се за одлуку коју

променљиву одабрати може користити хеуристика степена (енг. degree heuristic). Ова хеуристика бира променљиву која учествује у већем броју ограничења у којим постоје променљиве којима није додељена вредност. На тај начин се покушава смањити степен гранања при будућим доделама.

Када је променљива одабрана од утицаја може бити којим редом ће јој вредности из скупа дозвољених вредности бити додељиване. У случају да вредности производе неконзистентно стање одмах или у будућности прелази се на друге вредности, међутим, ако је додела конзистентна постоји могућност да решење буде пронађено тј. да се претрага неће вратити назад до ове променљиве. Зато се при избору којим редом се вредности додељују користи хеуристика која бира вредност која најмање ограничава скупове дозвољених вредности других променљивих (енг. least constraining value). На тај начин се покушава фаворизовати проналажење решење.

Даља унапређења могу укључити коришћење посебних алгоритама за задовољавање неких типова ограничења на пример ограничења AllDifferent. Могуће је увести повратне скокове и учење ограничења. За ефикасније решавање неких проблема могуће је искористити неке особине њихове структуре на пример ако је граф конзистентности стабло, или искористити идеју декомпозиције. Детаљније о овим питањима може се видети у [8].

2.4 Псеудо-буловски решавачи

Како је у раду коришћено готово решење за свођење на SAT – псеудо-буловски решавачи, укратко ће бити описано шта је то псеудо-буловски проблем и псеудо-буловски решавач.

Псеудо-буловски проблем је проблем испитивања задовољивости скупа

ограничења код ког све променљиве узимају вредности из скупа $\{0,1\}$, а ограничења су задата у облику једначина и неједначина над променљивим. Ограничења могу бити и нелинеарна. Ако се захтева и налажење решења које које оптимизује одређену функцију циља ради се о оптимизационом псеудо-буловском проблему.

Псеудо-буловски проблеми се могу сматрати проширењем SAT проблема. Од ограничење облика $C_0 x_0 + ... + C_{n-1} x_{n-1} \geq C_n$ клауза се може добити ако се вредности свих коефицијената поставе на 1. Псеудо-буловски проблем се може свести на скуп оваквих ограничења а решавање проблема представља налажење решења које задовољава дати скуп ограничења.

За решавање псеудо-буловских проблема проблем се преводи на SAT и користи се неки интерни SAT решавач. Постоје два приступа. Један је да се проблем директно преводи на SAT, док други приступ подразумева формирање хибридних решавача који у решавању користе и неке елементе из позадинске теорије и практично представљају проширење SAT решавача. У случају оптимизационих проблема обично се проблем интерно своди на проблем одлучивања на стандардан начин тако што се уводи питање тј. ново ограничење које испитује да ли је вредност функције циља мања или једнака некој константи. Оптимизација се спроводи методама претраге (итеративна, бинарна или њихова комбинација) за што мањом вредношћу ове константе (у случају проблема минимизације).

2.5 CPLEX

CPLEX је програмски пакет за решавање различитих врста оптимизационих проблема. Може се користити за решавање проблема линеарног програмирања, проблема мешовитог целобројног програмирања, проблема квадратног програмирања и проблема квадратно ограниченог програмирања [3]. Неки од

метода које садржи су симплекс метод за решавање проблема линеарног програмирања, симплекс метод и метод баријере за решавање проблема квадратног програмирања, метод гранања и одсецања комбинован са хеуристикама за решавање проблема целобројног односно мешовитог целобројног програмирања. Како је у раду пажња посвећена проблемима целобројног и мешовитог целобројног програмирања биће укратко описан само метод гранања и ограничавања.

2.5.1 Метод гранања и ограничавања

Оно што проблем целобројног програмирања разикује од осталих оптимизационих проблема је захтев да неке, могуће и све, променљиве узимају целобројне вредности. Претпоставка је да се надаље говори о целобројном линеарном програмирању.

Уклањањем целоборјног ограничења добија се релаксација полазног проблема која се може решити симплекс методом. У случају да је добијено решење целобројно алгоритам се завршава. У случају да решење није целобројно најпре се проналази раван раздвајања. Раван раздвајања представља додатно ограничење које се додаје линеарној релаксацији полазног проблема. Ово ограничење је такво да га нецелобројно оптимално решење неће задовољавати, али га задовољавају сва допустива целобројна решења. Након тога се решава измењени проблем. Ако је решење и даље нецелобројно онда се врши гранање.

Гранање подразумева да се простор допустивих решења дели на два или више делова. На тај начин се добијају два подпроблема која се затим решавају на описани начин. Овако се формира дрво подпроблема па се потпроблеми често називају и чворови стабла. Гранање се најчешће врши применом гранања по променљивој. То значи да се у нецелобројном решењу проналази променљива x_i

која има нецелобројну вредност $\bar{x}_i$ и добијају се два подпроблема у где се једном додаје ограничење $x_i \leq \bar{x}_i$ а другом ограничење $x_i \geq \bar{x}_i$. Приликом гранања врше се одсецања. Под претпоставком да се ради о проблему минимизације, доњу границу представља вредност функције циља линеарне релаксације. Горњу границу престаљва вредност функције циља за неко већ пронађено целобројно решење. Ако је у неком чвору доња граница већа од постојеће горње границе тај чвор и његови евентуални потомци могу бити елиминисани из разматрања.

Гранање стаје када је у скупу допустивих решења подпроблема остало само једно целобројно решење или када је доња граница једнака горњој граници неког потпроблема.

2.6 Генетски алгоритам

Генетски алгоритам је једна од метахеуристика која је инспирисана процесима који се одвијају у физици, биологији, технологији итд. Припада ужој групи алгоритама инспирисаних биолошким процесима [13]. Заснован је на природним процесима еволуције и природне селекције. Иако је првобитно био намењен симулирању ових процеса тј. симулирању промена популације под утицајем околине и генетских оператора показало се да може успешно бити примењен на решавање оптимизационих проблема.

Свака јединка у природи поседује свој генетски материјал, скуп информација које одређују особине јединке. У зависности од особина које поседују јединке су у одређеном степену прилагођене условима средине у којој живе. Према теорији еволуције, јединке одређене популације које су најприлагођеније условима околине преживљавају и свој генетски материјал преносе у наредну генерацију формирајући нову популацију. Нове особине у популацији се могу јавити и појавом мутација, недетерминистичких промена у генетском материјалу, које могу

имати најразличитије узроке.

Генетски алгоритам се заснива на сличним поставкама. У генетском алгоритму се сваком решењу из простора допустивих решења се придружује одређени низ симбола који се назива код решења. Уобичајен назив за код решења у контексту генетског алгоритма је хромозом (део генетског материјала у биолошком смислу). Након што је решењу придружен његов код могуће је израчунати прилагођеност решења која зависи од природе проблема. За проблеме математичке оптимизације који имају јасно дефинисану функцију циља обично се прилагођеност рачуна као вредност функције циља за дато решење (јединку). За друге врсте проблема код којих функција циља није дефинисана потребно је посебно дефинисати функцију прилагођености. При извршавању генетског алгоритма одржава се популација решења. Над том популацијом се примењују генетски и еволуциони оператори.

Прво питање које треба решити је политика замене генерација. Ово питање односи се на то како ће се формирати наредна генерација у итерацији алгоритма. Један приступ је да се целокупна следећа генерациј формира укрштањем селектованих јединки. Други приступ је сатационарна политика замене генерација која подразумева да се део тренутне популације преноси у наредну генерацију а други део се добија укрштањем селектованих јединки. Трећи приступ је елитизам. Елитизам подразумева да се одређени број најквалитетнијих јединки преноси у наредну генерацију.

Затим се на целу полазну популацију примењује оператор селекције. Оператор селекције на основу вредности функције прилагођености бира које јединке чија ће деца формирати наредну генерацију. Најпростији метод селекције би могао бити да се селектује одређен број јединки које имају највећу вредност функције прилагођености. Међутим, овакав приступ би могао довести до брзе конвергенције ка неком локалном минимуму па се ретко примењује. Постоји доста различитих метода селекције а често и неколико варијанти истог метода. Неке од најчешћих су рулетска и турнирска селекција.

Након примене оператора селекције над селектованим јединкама се примењује оператор укрштања. Оператор укрштања симулира природни процес репродукције. Бирају се две јединке из популације, које се називају родитељи, и са одређеном вероватноћом се врши њихово укрштање тј. размена генетског материјала. Резултат укрштања две јединке је једна или две нове јединке (деца) које наслеђују део генетског материјала од једног а део од другог родитеља. У зависности од репрезентације јединки могуће је применити различите начине укрштња. Међу најчешћим начинима укрштања је укрштање са једном тачком прекида, при чему се одабира тачка прекида па дете наслеђује део генетског материјала од једног родитеља до одабране тачке прекида а од тачке прекида дете наслеђује генетски материјал од другог родитеља. Могуће је користити и више тачака прекида. Присутно је и униформно укрштање које обично даје два детета при чему се ген првог родитеља преноси на прво дете са вероватноћом p а на друго дете са вероватноћом 1-p, дете које није одабрано наслеђује ген од другог родитеља.

Други генетски оператор који је део стандардног алгоритма је оператор мутације. Укрштањем се добијају јединке које би требало да су јако сличне јединкама претходне популације. Ако би се само примењивало укрштање могло би врло лако да дође преурањене конвергенције ка локалном оптимуму пошто би у неколико генерација могао да буде претражен целокупан део простора претраге који је доступан из иницијалне популације јер укрштање не поседује механизам за уношење нових информација у популацију. Ту на сцену ступа мутација. Мутацијом се у популацију уносе нове информације које имају за циљ да спрече конвергенцију ка локалном оптимуму и помогну у потрази за глобалним оптимумом. Ово је јако слично улози коју мутација има у природи где омогућава одржању генетске разноврсности. Мутација се најчешће примењује на јединку са одређеном малом вероватноћом тако што се одабира који ген ће бити промењен.

Критеријум заустављања служи да означи крај примене алгоритма. Критеријуми

заустављања могу бити различити а најчешће се примењују комбинације неколико критеријума заустављања. Један од најједноставнијих критеријума заустављања би мога да буде да је достигнут одређени број итерација, у контексту генетског алгоритма називају се још и генерацијама. Други услов би мога бити да буде да је пронађено решење које задовољава одређени унапред задате критеријуме. Наравно, могући су и други услови.

Псеудокод генетског алгоритма:

```
Resenje GA()
{
     Populacija p = inicijalizuj_populaciju();
     while(!kriterijum_zaustavljanja(p))
     {
          smena_generacija(p);
          selektuj_jedinke(p);
          ukrsti_jedinke(p);
          mutiraj_jedinke(p);
     }
     return najbolje(p);
}
```

2.7 Локална претрага

Локална претрага је метахеуристика која на основу неког датог допустивог решења оптимизационог проблема и дате дефиниције околине покушава да у околини датог решења прегледа сва допустива решења и пронађе неко решење са бољом вредношћу функције прилагођености. У случају да је такво решење пронађено тада кажемо да представља локални оптимум међу решењима у својој околини. Да

ли то решење представља и глобални оптимум не може се са сигурношћу тврдити. У следећој итерацији полази се од пронађеног решења и поново се покреће потрага у његовој околини. Када боље решење није могуће пронаћи, или је задовољен неки други кртеријум заустављања претрага се зауставља и тренутно решење се враћа као коначно.

Начин за добијање полазног решења није прописан па се такво решење може добити псеудослучајно, коришћењем неке друге (мета)хеуристике или неког похлепног алгоритма.

Околину је могуће дефинисати на различите начине. При дефинисању околине треба бити опрезан. Ако је околина дефинисана тако да обухвата велики број решења извршавање алгоритма може бити превише временски захтевно. Ако је дефинисана тако да обухвата мали број решења добијено решење може бити лошег квалитета. Треба напоменути да се некада због побољшања ефикасности не врши прегледање свих решења у околини тренутног решења већ се претрага околине врши до проналаска првог бољег решења након чега се преналази у нову околину.

Поред заустављања у случају да није могуће пронаћи боље решење ко критеријум заустављања могуће је користити достизање одређеног броја итерација. Алгоритам се може зауставити и у случају да пронађено решење задовољава одређене задате критеријуме. Најчешће се користи комбинација неколико критеријума заустављања.

Локална претрага је изузетно зависна од избора полазног допустивог решења. Врло лако се може десити да резултат алгоритма буде локални оптимум. Зато се локална претрага не користи често у свом чистом облику већ су присутне различите модификације локалне претраге као што су итеративна локална претрага, табу претраживање и друге или се локална претрага комбинује са неким другим метахеуристикама – такозвана хибридизација.

Псеудокод локалне претраге:

```
Resenje LP()
{
     Resenje trenutno = inicijalizuj_resenje();
     do
     {
          Resenje sledece = trenutno.sledece();
          if(sledece.f < trenutno.f)
          {
               trenutno = sledece;
          }
     }
     while(!kriterijum_zaustavljanja)
     return trenutno;
}
```

2.8 Хибридизација генетског алгоритма и локалне претраге

Хибридизација метахеуристика је значајна техника којом се покушавају превазићи недостаци појединих метахеуристика. Тако на пример метахеуристике базиране на једном решењу могу теже претраживати велике просторе претраге, док популационе метахеуристике можда неће довољно детаљно претражити одређене делове простора претраге. Због тога је уобичајено комбиновање популационих и метахеуристика базираних на једном решењу.

Алгоритам настао као хибридизација локалне претраге и генетског алгоритма тако што се локална претрага примењује као оператор унутар генетског алгоритма назива се и меметски алгоритам [12]. Локална претрага као чист алгоритам има значајне недостатке. Ипак, коришћење локалне претраге у оквиру генетског

алгоритма, који због тога што припада групи популационих метахеуристика лакше претражује различите делове простора допустивих решења, може допринети бржој конвергенцији ка оптималном решењу док је при томе вероватноћа конвергенције ка локалном оптимуму смањена.

Начини хибридизације могу бити различити. Један је да се локална претрага са одређеном вероватноћом примењује на јединке популације након примене генетских оператора. Локална претрага се може примењивати и након испуњавања неког другог критеријума као што се наводи. Могуће је локалном претрагом иницијализовати популацију, али постоји могућност да би се тако добила превише једнолична прва генерација. Могуће је локалну претрагу применити на решење добијено као резултат рада генетског алгоритма, али на тај начин се врши само фино подешавање добијеног решења, тако да добијена побољшања не морају бити значајна.

Псеудокод хибридизованог алгоритма:

```
Resenje hibrid()
{
     Populacija p = inicijalizuj_populaciju();
     while(!kriterijum_zaustavljanja(p))
     {
          smena_generacija(p);
          selektuj_jedinke(p);
          ukrsti_jedinke(p);
          mutiraj_jedinke(p);
          lokalna_pretraga(p);
     }
     return najbolje(p);
}
```

3. Начини решавања двонивоске локације постројења са неограниченим капацитетима (TSUFLP)

Проблеми локација и алокације су један тип проблема комбинаторне оптимизиције и од изузетног су значаја у области управљања ланцима снабдевања. У ове проблеме спадају различити проблеми у којима је потребно успоставити локације неких ресурса / постројења да би се задовољиле одређене потребе тј. испунили одређени циљеви. На успостављање постројења могу утицати различити фактори као што су удаљеност од чворова чију које ће опслуживати, капацитет постројења, цена успостављања постројења и други. Обично је при одлучивању потребно минимизовати трошкове снабдевања или максимизовати профит, или се може користити неки други критеријум оптимизације. Дакле, циљ је одредити оптималну расподелу постројења у складу са захтеваним ограничењима.

3.1 Проблем двонивоске локације постројења са неограниченим капацитетима (TSUFLP)

У случају када постоји једна централна јединица не дозвољава се увек слободан приступ крајњим корисницима тој централној јединици из различитих разлога [7]. Такав приступ уобичајен је у телекомуникационим мрежама где се јавља проблем оптимизације распореда више нивоа концентратора протока података. У таквим системима крајњи корисник се повезује на концентраторе првог нивоа, који се затим повезују на концентраторе виших нивоа и тако даље. Тек се концентратори највишег нивоа повезују на централне јединице.

Проблем двонивоске локације постројења са неограниченим капацитетом је настао као један одговор на овакве потребе. У овом проблему постоје два нивоа концентратора. Одређен број концентратора првог и другог нивоа се успоставља на подскуповима скупова дозвољених локација. Терминали, или крајњи

корисници, повезују се на концентраторе првог нивоа, који се повезују на концентраторе другог нивоа, а ови се вежу на централну јединицу. Усвостављње концентратора и успостављање везе између терминала и концентратора првог нивоа и концентратора првог са концентраторима другог нивоа имају фиксне цене. Циљ је минимизовати цену успостављања мреже.

О овом проблему, његовим варијантама, као и упути на друге проблеме вишенивоске локације постројења може се видети у литератури [7].

3.2 Математички модел

Као математички модел проблема коришћена је формулација наведена у литератури [7].

Дат је скуп локација терминала *N*, скуп дозвољених локација за концентраторе првог нивоа *M* и скуп дозвољених локација за концентраторе другог нивоа *K*. Додељивање терминала концентратору првог нивоа има цену $C_{ij}, i \in N, j \in M$. Цена успостављања концентратора првог нивоа на локацији j и његава додела концентратору другог нивоа на локацији k је $B_{jk}, j \in M, k \in K$. Цена успостављања концентратора другог нивоа је $F_k, k \in K$. Потребно је успоставити концентраторе првог нивоа и њима доделити терминале и успоставити концентраторе другог нивоа и њима доделити успостављене концентраторе првог нивоа тако да укупна цена успостављња мреже буде минимална.

Математички модел:

$$(1)\quad min \sum_{i \in N} \sum_{j \in M} C_{ij} x_{ij} + \sum_{j \in M} \sum_{k \in K} B_{jk} y_{jk} + \sum_{k \in K} F_k z_k$$

при ограничењима:

$$(2)\quad \sum_{j \in M} x_{ij} = 1, \text{ за свако } i \in N$$
$$(3)\quad x_{ij} \leq \sum_{k \in K} y_{jk}, \text{ за свако } i \in N, \quad j \in M$$
$$(4)\quad y_{jk} \leq z_k, \text{ за свако } j \in M, \quad k \in K$$
$$(5)\quad \sum_{k \in K} y_{jk} \leq 1, \text{ за свако } j \in M$$
$$(6)\quad x_{ij} \in \{0,1\}, \text{ за свако } i \in N, \quad j \in M$$
$$(7)\quad y_{jk} \in \{0,1\}, \text{ за свако } j \in M, \quad k \in K$$
$$(8)\quad z_k \in \{0,1\}, \text{ за свако } k \in K$$

(1) преставља функцију циља која минимизује укупну цену успостављања мреже тј. збир цене доделе терминала концентраторима првог нивоа (први део), цене успостављања концентратора нивоа и њихове доделе концентраторима другог нивоа(други део) и цене успостављања концентратора другог нивоа. (2) је ограничење које обезбеђује да је терминал додељен тачно једном концентратору првог нивоа. (3) је ограничење које обезбеђује да је терминал додељен само концентратору првог нивоа који је успостављен. Ограничења (4) и (5) обезбеђују да је успостављени концентратор првог нивоа додељен једном концентратору другог нивоа. Остала ограничења су ограничења над вредностима променљивих.

Описани проблем је NP-тежак [7].

3.3 Решавање свођењем на SAT

За решавање свођењем на SAT коришћен је псеудо-буловски решавач sat4j. Уобичајено псеудо-буловски решавачи улаз прихватају у opb формату који је установљен на такмичењима псеудобуловских решавача.

У овом формату је препорука у првом коментару, коментари се означвају са *,

навести број променљивих и ограничења. У првој линији описа проблема се наводи функција циља означена са min : , ако се ради о проблему псеудобуловске оптимизације, у супротном овај део се изиставља. Затим се у наредним линијама које описују проблем наводе ограничења која треба да задовољава решење. Ограничења се састоје од леве стране која представља збир производа целобројних константи и променљивих, затим следи оператор поређења при чему су дозовољени само веће или једнако и једнако као оператори поређења, и десна страна ограничења која се састоји само из целобројне константе. Свака линија којом се описује проблем завршава се са ; . Детаљније о формату се може наћи на [9].

Ако је инстанца дата следећим описом, где су у првој линији дате димензије проблема n, m, k, затим матрице цена C, B и вектор цена F, како су обисани у математичком моделу проблема:

```
5 3 2

12 22 18
14 19 22
20 31 13
22 21 2
17 24 9

29 12
28 31
21 13

16 30
```

Оваква инстанца се кодира у opb облику на следећи начин:

```
* #variable= 23 #constraint= 29
min : 12 x1 22 x2 18 x3 14 x4 19 x5 22 x6 20 x7 31 x8 13 x9
22 x10 21 x11 2 x12 17 x13 14 x14 9 x15 29 x16 12 x17 28
```

```
x18 31 x19 21 x20 13 x21 16 x22 30 x23;
1 x1 1 x2 1 x3 = 1;
1 x4 1 x5 1 x6 = 1;
1 x7 1 x8 1 x9 = 1;
1 x10 1 x11 1 x12 = 1;
1 x13 1 x14 1 x15 = 1;
1 x16 1 x17 -1 x1 >= 0;
1 x18 1 x19 -1 x2 >= 0;
1 x20 1 x 21 -1 x3 >= 0;
1 x16 1 x17 -1 x4 >= 0;
1 x18 1 x19 -1 x5 >= 0;
1 x20 1 x 21-1 x6 >= 0;
1 x16 1 x17 -1 x7 >= 0;
1 x18 1 x19 -1 x8 >= 0;
1 x20 1 x 21 -1 x9 >= 0;
1 x16 1 x17 -1 x10 >= 0;
1 x18 1 x19 -1 x11 >= 0;
1 x20 1 x 21 -1 x12 >= 0;
1 x16 1 x17 -1 x13 >= 0;
1 x18 1 x19 -1 x14 >= 0;
1 x20 1 x 21 -1 x15 >= 0;
1 x22 -1 x16 >= 0;
1 x23 -1 x17 >= 0;
1 x22 -1 x18 >= 0;
1 x23 -1 x19 >= 0;
1 x22 -1 x20 >= 0;
1 x33 -1 x21 >= 0;
-1 x16 -1 x17 >= 0;
-1 x18 -1 x19 >= 0;
-1 x20 -1 x21 >= 0;
```

Описани проблем спада у групу оптимизационих псеудо-буловских проблема. Изабрани решавач подржава решавање ове групе проблема тако је решавање подразумевало само записивање проблема на opb формату. Кодирање је вршено директним превођењем описаног математичког модела на opb језик.

Оно о чему је посебно било потребно водити рачуна је да су у проблему коефицијенту у функцији циља цене тј. реални бројеви, а решавач не подржава коришћење реалних бројева. При решавању су коришћена два приступа, први да се вредности коефициената заокруже, и други да се ради са одређеном тачношћу при чему се коефицијенти множе.

3.4 Решавање свођењем на SMT

За решавање свођењем на SMT коришћен је SMT решавач yices. Пошто је овај решавач могуће користити само за испитивање задовољивости било је потребно дати оптимизациони проблем превости на проблем испитивања задовољивости. То је учињено увођењем додатне променљиве f и додавањем ограничења да је уведена променљива једнака вредности функције циља.

Затим је разматрано које вредности функција циља може узети. Теоријски минимум је 0, јер су у питању цене успостављања постројења које морају бити позитивне, а функција циља представља збир цена. Теоријски максимум зависи од максималне вредности цена у инстанци, о томе је вођено рачуна при формирању инстанци. Бинарном претрагом је вршена потрага за оптималном вредношћу функције циља у интервалу који је ограничен теоријским минимумом и теоријским максимумом. У сваком корају бинарне претраге формирана је проблем одлучивања од полазног оптимизационог проблема додавањем ограничења $f \leq c$, где је f уведена променљива а c нека константа из интервала добијена на основу бинарне претраге.

Инстанце су записиване у формату који је познат као yices језик пре свега зато што је у овом формату могуће истовремено користити и релне и целе бројеве тј. могуће је користити реалне коефициенте уз целобројне променљиве. Стандардни формат за SMT решаваче, SMT-LIB не подржава ову могућност. У сваком кораку бинарне претраге је фајл записан на овом језику модификован само мењањем граница за вредност функције циља.

Фајл записан на yices језику представља низ команди које интерпретира yices. Команде се записују у, за SMT решаваче уобичајеној, LIPS нотацији. Костур чине команде за дефинисање објеката нпр. функција, термова, типова. Затим се над тим скупом објеката формирају формуле које се одговарајућим командама пуне у контекст (assert) и затим се проверава задовољивост контекста (check). Ако је контекст задовољив могуће је наћи вредности термова у том контексту.

Уграђени типови су bool, int, real, bitvector. Могуће је дефинисати нове типове. Подржана је линеарна целобројна и линеарна реална аритметика тј. подржано је проверавање задовољивости формула без квантификатора ових теорија. Ово је олакшало кодирање на yices језику пошто је кодирање пратило математички модел.

SMT-LIB 2 запис се у погледу кодирања наведеног проблема не разликује много од yices језика, осим у погледу већ наведених разлика и различитих назива команди које имају еквивалентну функцију (нпр. у SMT-LIB постоји команда check-sat, док код yices check има исту функцију), пошто подржава испитивање задовољивости формула линеарне аритметике.

За инстанцу описану у 3.1.2 кодирање у yices формату би било следеће:

```
(f::int)
(x1::int) (x2::int) (x3::int) (x4::int) (x5::int) (x6::int)
(x7::int) (x8::int) (x9::int) (x10::int) (x11::int)
```

```
(x12::int) (x13::int) (x14::int) (x15::int) (x16::int)
(x17::int) (x18::int) (x19::int) (x20::int) (x21::int)
(x22::int) (x23::int)
(assert
(and
(= f (+ (* x1 12) (* x2 22) (* x3 18) (* x4 14) (* x5 19)
(* x6 22) (* x7 20) (* x8 31) (* x9 13) (* x10 22) (* x11
21) (* x12 2) (* x13 17) (* x14 24) (* x15 9) (* x16 29) (*
x17 12) (* x18 28) (* x19 31) (* x20 21) (* x21 13) (* x22
16) (* x23 30)))
(>= f 0)
(<= f 300)
(and (>= x1 0) (<= x1 1))
(and (>= x2 0) (<= x2 1))
(and (>= x3 0) (<= x3 1))
(and (>= x4 0) (<= x4 1))
(and (>= x5 0) (<= x5 1))
(and (>= x6 0) (<= x6 1))
(and (>= x7 0) (<= x7 1))
(and (>= x8 0) (<= x8 1))
(and (>= x9 0) (<= x9 1))
(and (>= x10 0) (<= x10 1))
(and (>= x11 0) (<= x11 1))
(and (>= x12 0) (<= x12 1))
(and (>= x13 0) (<= x13 1))
(and (>= x14 0) (<= x14 1))
(and (>= x15 0) (<= x15 1))
(and (>= x16 0) (<= x16 1))
(and (>= x17 0) (<= x17 1))
(and (>= x18 0) (<= x18 1))
(and (>= x19 0) (<= x19 1))
```

```
(and (>= x20 0) (<= x20 1))
(and (>= x21 0) (<= x21 1))
(and (>= x22 0) (<= x22 1))
(and (>= x23 0) (<= x23 1))
(= 1 (+ x1 x2 x3))
(= 1 (+ x4 x5 x6))
(= 1 (+ x7 x8 x9))
(= 1 (+ x10 x11 x12))
(= 1 (+ x13 x14 x15))
(<= x1 (+ x16 x17))
(<= x2 (+ x18 x19))
(<= x3 (+ x20 x21))
(<= x4 (+ x16 x17))
(<= x5 (+ x18 x19))
(<= x6 (+ x20 x21))
(<= x7 (+ x16 x17))
(<= x8 (+ x18 x19))
(<= x9 (+ x20 x21))
(<= x10 (+ x16 x17))
(<= x11 (+ x18 x19))
(<= x12 (+ x20 x21))
(<= x13 (+ x16 x17))
(<= x14 (+ x18 x19))
(<= x15 (+ x20 x21))
(<= x16 x22)
(<= x17 x23)
(<= x18 x22)
(<= x19 x23)
(<= x20 x22)
(<= x21 x23)
(<= (+ x16 x17) 1)
```

```
(<= (+ x18 x19) 1)
(<= (+ x20 x21) 1)
)
)
(set-timeout 600)
(check)
(eval f)
```

3.5 Решавање свођењем на CSP

За решавање свођењем на CSP коришћен је решавач minion. Овај решавач подржава решавање оптимизационих проблема тако да је проблем требало само записати у улазном формату за minion. Оно о чему је требало водити рачуна, као и у неким претходним случајевима, је да није подржан рад са реалним бројевима те је проблем било потребно превести у целобројни облик. Кодирање на minion језик пратило је математички модел уз незнатна прилагођавања.

На почетку minion фајла наводи се ознака верзије. Затим се наводе делови у којима се уводе променљиве проблема, ограничења и део у ком се спецификује начин претраге и излаз. Проблем се описује записивањем ограничења користећи уграђене предикате које подржава minion. На крају се наводи ознака за крај фајла након које се сав преостали текст игнорише.

Променљиве могу бити BOOL типа, то су променљиве које могу имати само вредности из скупа $\{0,1\}$, DISCRETE и BOUND типа који представљју променљиве које узимају вредности из одређеног интервала, и SPARSEBOUND типа који представља променљиве које узимају вредности из неког наведеног скупа вредности нпр. $\{1,5,10,50,100,500,1000\}$

Занимљиво је да minion нема предикат који пореди једнакост суме производа константи и променљивих са целобројном константом. За ову примену мора се користити комбинација предиката који пореде да ли је сума већ једнака или једнака и мања или једнака од целобројне константе.

Такође, minion не подржава минимизацију суме која би представља функцију циља већ искључиво минимизовање вредности променљиве. Зато је било неопходно увести додатну променљиву која би чувала вредност функције циља.

Додатно о формату и доступним ограничењима може се видети у [10].

За инстанцу описану у 3.1.2 кодирање у minion формату би било следеће:

```
MINION 3
**VARIABLES**
BOUND f {0..300}
BOOL x1 BOOL x2 BOOL x3 BOOL x4 BOOL x5 BOOL x6 BOOL x7
BOOL x8 BOOL x9 BOOL x10 BOOL x11 BOOL x12 BOOL x13 BOOL
x14 BOOL x15 BOOL x16 BOOL x17 BOOL x18 BOOL x19 BOOL x20
BOOL x21 BOOL x22 BOOL x23
**CONSTRAINTS**
weightedsumleq([12, 22, 18, 14, 19, 22, 20, 31, 13, 22 21
2, 17, 24, 9, 29, 12, 28, 31, 21, 13, 16, 30, -1], [x1, x2,
x3, x4, x5, x6, x7, x8, x9, x10, x11, x12, x13, x14, x15,
x16, x17, x18, x19, x20, x21, x22, x23, f], 0)
weightedsumgeq([12, 22, 18, 14, 19, 22, 20, 31, 13, 22 21
2, 17, 24, 9, 29, 12, 28, 31, 21, 13, 16, 30, -1], [x1, x2,
x3, x4, x5, x6, x7, x8, x9, x10, x11, x12, x13, x14, x15,
x16, x17, x18, x19, x20, x21, x22, x23, f], 0)
sumleq([x1, x2, x3], 1)
sumgeq([x1, x2, x3], 1)
sumleq([x4, x5, x6], 1)
```

```
sumgeq([x4, x5, x6], 1)
sumleq([x7, x8, x9], 1)
sumgeq([x7, x8, x9], 1)
sumleq([x10, x11, x12], 1)
sumgeq([x10, x11, x12], 1)
sumleq([x13, x14, x15], 1)
sumgeq([x13, x14, x15], 1)
weightedsumgeq([1, 1, -1], [x16, x17, x1], 0)
weightedsumgeq([1, 1, -1], [x18, x19, x2], 0)
weightedsumgeq([1, 1, -1], [x20, x21, x3], 0)
weightedsumgeq([1, 1, -1], [x16, x17, x4], 0)
weightedsumgeq([1, 1, -1], [x18, x19, x5], 0)
weightedsumgeq([1, 1, -1], [x20, x21, x6], 0)
weightedsumgeq([1, 1, -1], [x16, x17, x7, 0)
weightedsumgeq([1, 1, -1], [x18, x19, x8], 0)
weightedsumgeq([1, 1, -1], [x20, x21, x9], 0)
weightedsumgeq([1, 1, -1], [x16, x17, x10], 0)
weightedsumgeq([1, 1, -1], [x18, x19, x11], 0)
weightedsumgeq([1, 1, -1], [x20, x21, x12], 0)
weightedsumgeq([1, 1, -1], [x16, x17, x13], 0)
weightedsumgeq([1, 1, -1], [x18, x19, x14], 0)
weightedsumgeq([1, 1, -1], [x20, x21, x15], 0)
weightedsumleq([1, -1], [x16, x22], 0)
weightedsumleq([1, -1], [x17, x23], 0)
weightedsumleq([1, -1], [x18, x21], 0)
weightedsumleq([1, -1], [x19, x23], 0)
weightedsumleq([1, -1], [x20, x22], 0)
weightedsumleq([1, -1], [x21, x23], 0)
sumleq([x16, x17], 1)
sumleq([x18, x19], 1)
sumleq([x20, x21], 1)
```

```
**SEARCH**
MINIMISING f
PRINT [[f]]
**EOF**
```

3.6 Решавање коришћењем CPLEX пакета

CPLEX је потпуно прилагођен решавању оптимизационих проблема тако да у овом случају није било потребно вршити никаква прилагођавања. Проблем је записан у lp формату и тако предат CPLEX-у.

3.7 Решавање коришћењем генетског алгоритма

Јединке су представљене бинарним генетским кодом. Код се састоји из два дела. Први део је део димензије *m* где сваки од *m* битова представља концентратор првог нивоа. Ако је на месту *j* 0 то значи да је *j*-ти концентратор укључен, ако је на месту *j* 0 то значи да је *j*-ти концентратор искључен. Други део је димензије *k*, ознаке су исте као у првом делу. Пошто се ради о проблему са неограниченим капацитетама није потребно представљати терминале већ се они могу похлепним приступом додељивати концентратору првог нивоа за који је цена доделе минимална. На сличан начин се концентратори првог додељују концентраторима другог нивоа.

Популација се иницијализује тако што се свака јединка иницијализује на случајан начин. Случајно се одређује колико ће концентратора бити укључено на сваком од нивоа, а затим се случајно одређује који ће концентратори бити укључени.

Као критеријум заустављања је коришћено је да је максималан број итерација 100

и да је број итерација без побољшања тренутно најбољег решења мањи од 20.

Смена генерација се врши тако што се бира турнирском селекцијом 50% јединки које ће учествовати у репродукцији и оне се преносе у наредну генерацију. Остале јединке се добијају репродукцијом.

Укрштање се обавља са две тачке укрштања, по једна у сваком делу генетског кода. Свака тачка укрштања се бира на случајан начин а дете наслеђује први део података од првог а други део од другог родитеља.

Мутацијом се случајно одређује бит ком ће бити промењена вредност тј. на случајан начин се један концентратор или укључује или искључује у односу на претходно решење.

Као решење алгоритм враћа најбоље решење последње популације.

3.8 Решавање коришћењем локалне претраге

Репрезентација решења је идентична репрезентацији јединке описаној у 3.7. Почетно решење се добија на исти начин као јединке за популацију генетског алгоритма. Критеријум заустављања је такође исти као за генетски алгоритам.

Околину решења чине сва решења која се од датог разликују у вредности макар једног бита репрезентације. Које од решења из околине ће бити прегледано одређује се на случајан начин. Не прегледају се сва решења из околине већ се прелази у прво пронађено решење са бољом вредношћу функције циља.

3.9 Решавање коришћењем хибридизације генетског алгоритма и локалне претраге

Хибридизација подразумева да се локална претрага додаје као оператор описаног генетског алгоритма.

Најпре се иницијализује популација као што је описано у 3.7. Користи се исти критеријум заустављања као у 3.7. Редом се примењују селекција, укрштање и мутација. Након тога се са вероватноћом 0.2 за сваку јединку нове популације покреће локална претрага при чему се јединка над којом је покренута локална претрага узима као иницијално решење у алгоритму локалне претраге описаном у 3.8. Ако локална претрага нађе бољу јединку та јединка замењује јединку популације за коју је покренута локална претрага.

4. Резултати тестирања

У овом поглављу описано је на који начин су добијене инстанце које су решаване као и агрегирани резултати метода. Детаљни резултати извршавања метода приказани су у додатку.

4.1 Инстанце

Најпре је идеја била да се методе тестирају на 50 инстанци чије би димензије биле равномерно распоређене од n=10, m=8, k=4 до n=500, m=400, k=200. Максимална цена код ових инстанци била је ограничена на 100. Цене су генерисане као случајни бројеви из интервала $[0,100]$. Решавање је обављано на кућном рачунару ограничених ресурса.

При решавању прве групе од 35 оваквих инстанци CPLEX се показао релативно добро, без обзира на ограничене ресурсе. У свега неколико случајева је пробио временски лимит који је постављан на 1800 секунди. Меморијски лимит није постављан али је до прекокачења доступне меморије дошло у само два случаја. Остале егзактне методе показале су се врло лоше. Одликовало их је пробијање временског лимита већ за првих 10 инстанци, нпр. minion је пробијао временски лимит почев од друге инстанце, и јако брзо потпуно искоришћење меморије. Резултати су били лоши у целобројном случају, још лошији у случају када је рађено са тачношћу 10^{-3} . У овој првој фази је за свођење на SAT коришћен решавач sugar који се показао неупотребљивим због великих меморијских захтева и велике КНФ добијене након превођења на SAT која је већ за прву инстанцу била неколико стотина мегабајта.

Због тога су направљене нове инстанце са целобројним ценама, sugar је замењен са sat4j, а инстанце су решаване егзактним методама на jason.matf.bg.ac.rs . CPLEX

је коришћена на полазном рачунару. Нове инстанце су формиране у три групе за које су границе цена постављене на 10, 20 и 30. У свакој групи формирано је по 5 инстанци са границама од n=10, m=8, k=4 до n=50, m=40, k=20. За сваку од ових инстанци формиране су 3 верзије. Временско ограничење је постављено на 600 секунди. Добијени резултати су приказани у следећем делу.

4.2 Добијени подаци

Табеле са подацима о извршавању на појединачним инстанцама дате су у додатку. Овде су приказани агрегирани подаци.

График 1 – збирно време извршавања сваке методе за инстанце са различитим границама цена

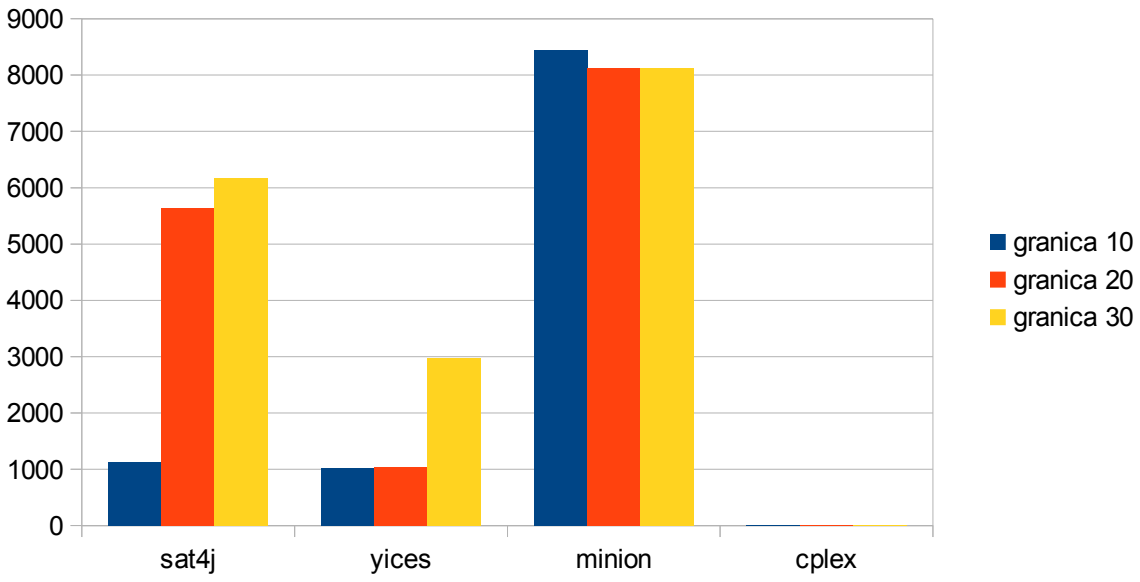


График 2 – број достигнутух оптималних решења за различите егзактне методе за инстанце са различитим границама цена

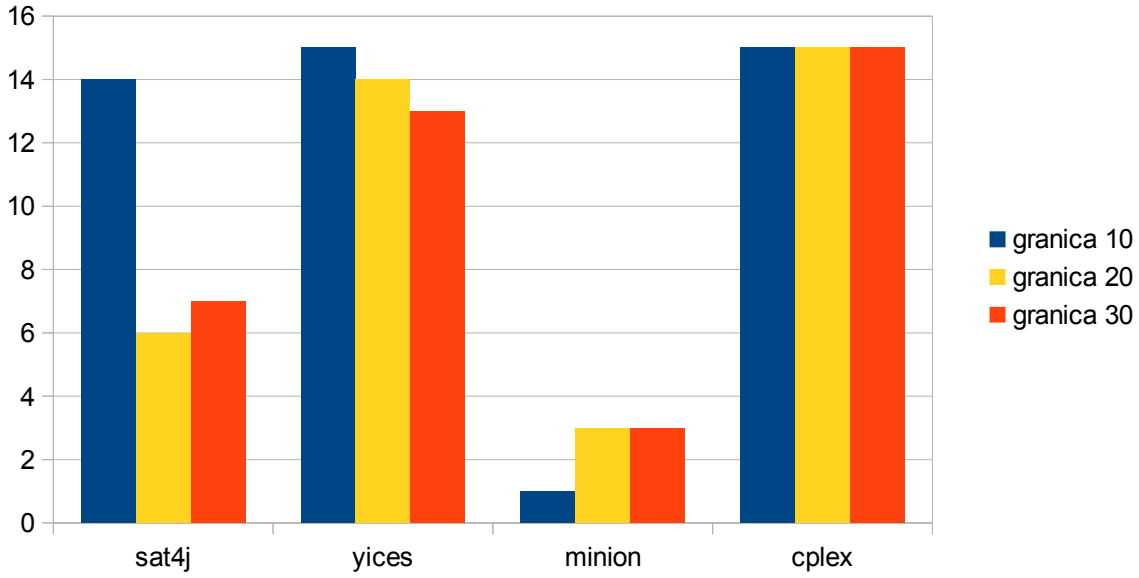


График 3 – збирно време извршавања за CPLEX и метахеуристичке методе з инстанце са различитим границама цена

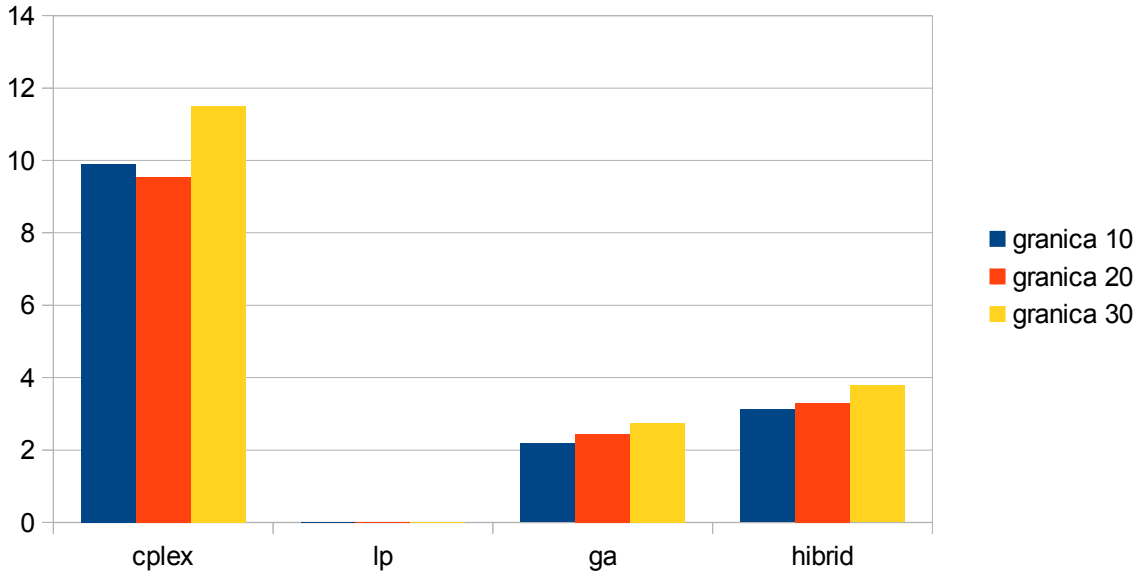
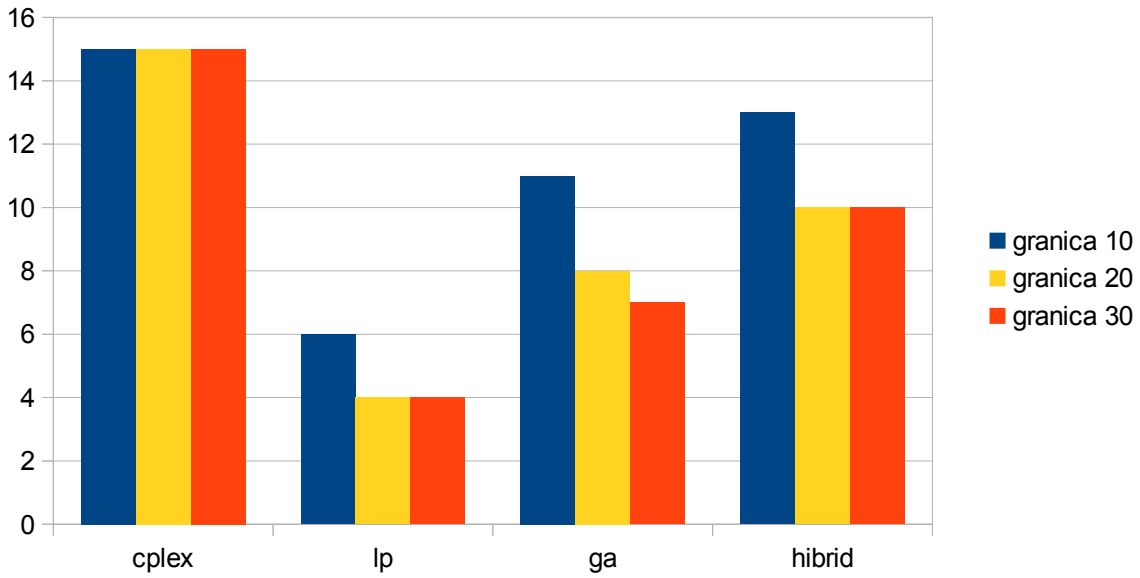


График 4 – број достигнутух оптималних решења за CPLEX и различите хеуристичке методе за инстанце са различитим границама цена



5. Закључак

CPLEX се показао изузетно ефикасним на инстанцама малих димензија где је изузетно брзо достизао оптимална решења, док су остале егзактне методе показале одређене недостатке, па се дешавало не само да је време извршавања дуго него и да оптимална решења нису достугнута. Ово се посебно односи на minion решавач, који се показао готово неупотребљивим за решавање овог проблема, чак и на малим инстанцама. SMT решавач yices се показао најбоље, док је sat4j био осредњи. Узевши у обзир да су егзактне методе поређене на инстанцама израдито малих димензија, може се закључити да резултати показују да је најпогоднија егзактна метода за решавање проблема двонивоске локације постројења са неограниченим капацитетима CPLEX док остале методе имају ограничену употребљивост.

Резултати извршашавања хеуристичких метода су приказани само за инстанце малих димензија. Најлошије се показала локална претрага што указује да је њену имплементацију потребно додатно побољшати. Генетски алгоритам је други, док се најбоље показао хибридни приступ. Ипак, како је CPLEX на овим примерима показао изузетно добро корисност хеуристичких метода би се најбоље видела испитивањем на примерима који представљају изазов за егзактно решавање, на пример на скупу инстанци великих димензија на којима је првобитно било замишљено тестирати све методе. Ово је остављено за даљи рад.

Додатак

Потпуни резултати извршавања свих метода.

Табела 1 – CPLEX резултати

инстанца	**n**	**m**	**k**	**граница цена**	**време извршавања**	**функција циља**	**оптималност**
instanca_10_v1_0	10	8	4	10	0.531	19	оптимално
instanca_10_v1_1	20	16	8	10	0.547	13	оптимално
instanca_10_v1_2	30	24	12	10	0.625	9	оптимално
instanca_10_v1_3	40	32	16	10	0.656	10	оптимално
instanca_10_v1_4	50	40	20	10	0.641	6	оптимално
instanca_10_v2_0	10	8	4	10	0.546	32	оптимално
instanca_10_v2_1	20	16	8	10	0.563	15	оптимално
instanca_10_v2_2	30	24	12	10	0.578	12	оптимално
instanca_10_v2_3	40	32	16	10	1.016	17	оптимално
instanca_10_v2_4	50	40	20	10	0.64	5	оптимално
instanca_10_v3_0	10	8	4	10	0.563	28	оптимално
instanca_10_v3_1	20	16	8	10	0.578	15	оптимално
instanca_10_v3_2	30	24	12	10	0.562	14	оптимално
instanca_10_v3_3	40	32	16	10	1.204	22	оптимално
instanca_10_v3_4	50	40	20	10	0.656	4	оптимално

instanca_20_v1_0	10	8	4	20	0.563	64	оптимално
instanca_20_v1_1	20	16	8	20	0.594	44	оптимално
instanca_20_v1_2	30	24	12	20	0.609	32	оптимално
instanca_20_v1_3	40	32	16	20	0.656	39	оптимално
instanca_20_v1_4	50	40	20	20	0.703	37	оптимално
instanca_20_v2_0	10	8	4	20	0.547	45	оптимално
instanca_20_v2_1	20	16	8	20	0.609	55	оптимално
instanca_20_v2_2	30	24	12	20	0.61	41	оптимално
instanca_20_v2_3	40	32	16	20	0.656	50	оптимално
instanca_20_v2_4	50	40	20	20	0.687	42	оптимално
instanca_20_v3_0	10	8	4	20	0.563	48	оптимално
instanca_20_v3_1	20	16	8	20	0.578	46	оптимално
instanca_20_v3_2	30	24	12	20	0.657	56	оптимално
instanca_20_v3_3	40	32	16	20	0.625	52	оптимално
instanca_20_v3_4	50	40	20	20	0.89	42	оптимално
instanca_30_v1_0	10	8	4	30	0.547	61	оптимално
instanca_30_v1_1	20	16	8	30	0.609	85	оптимално
instanca_30_v1_2	30	24	12	30	0.61	46	оптимално
instanca_30_v1_3	40	32	16	30	0.672	50	оптимално

instanca_30_v1_4	50	40	20	30	0.765	74	оптимално
instanca_30_v2_0	10	8	4	30	0.563	47	оптимално
instanca_30_v2_1	20	16	8	30	0.609	64	оптимално
instanca_30_v2_2	30	24	12	30	0.75	80	оптимално
instanca_30_v2_3	40	32	16	30	0.64	59	оптимално
instanca_30_v2_4	50	40	20	30	0.766	88	оптимално
instanca_30_v3_0	10	8	4	30	0.578	58	оптимално
instanca_30_v3_1	20	16	8	30	0.594	71	оптимално
instanca_30_v3_2	30	24	12	30	0.609	72	оптимално
instanca_30_v3_3	40	32	16	30	1.141	98	оптимално
instanca_30_v3_4	50	40	20	30	1.156	71	оптимално

Табела 2 – minion резултати

инстанца	**n**	**m**	**k**	**граница цена**	**време извршавања**	**функција циља**	**оптималност**
instanca_10_v1_0	10	8	4	10	45.37108	19	оптимално
instanca_10_v1_1	20	16	8	10	лимит	77	не
instanca_10_v1_2	30	24	12	10	лимит	99	не
instanca_10_v1_3	40	32	16	10	лимит	166	не
instanca_10_v1_4	50	40	20	10	лимит	228	не

instanca_10_v2_0	10	8	4	10	лимит	32	не (није стигао да докаже оптималност)
instanca_10_v2_1	20	16	8	10	лимит	80	не
instanca_10_v2_2	30	24	12	10	лимит	146	не
instanca_10_v2_3	40	32	16	10	лимит	133	не
instanca_10_v2_4	50	40	20	10	лимит	158	не
instanca_10_v3_0	10	8	4	10	лимит	28	не (није стигао да докаже оптималност)
instanca_10_v3_1	20	16	8	10	лимит	60	не
instanca_10_v3_2	30	24	12	10	лимит	109	не
instanca_10_v3_3	40	32	16	10	лимит	192	не
instanca_10_v3_4	50	40	20	10	лимит	194	не
instanca_20_v1_0	10	8	4	20	507.97094	64	оптимално
instanca_20_v1_1	20	16	8	20	лимит	132	не
instanca_20_v1_2	30	24	12	20	лимит	204	не
instanca_20_v1_3	40	32	16	20	лимит	349	не
instanca_20_v1_4	50	40	20	20	лимит	404	не
instanca_20_v2_0	10	8	4	20	16.79553	45	оптимално
instanca_20_v2_1	20	16	8	20	лимит	149	не
instanca_20	30	24	12	20	лимит	245	не

_v2_2							
instanca_20_v2_3	40	32	16	20	лимит	338	не
instanca_20_v2_4	50	40	20	20	лимит	473	не
instanca_20_v3_0	10	8	4	20	398.64732	48	оптимально
instanca_20_v3_1	20	16	8	20	лимит	133	не
instanca_20_v3_2	30	24	12	20	лимит	297	не
instanca_20_v3_3	40	32	16	20	лимит	373	не
instanca_20_v3_4	50	40	20	20	лимит	434	не
instanca_30_v1_0	10	8	4	30	163.39371	61	оптимально
instanca_30_v1_1	20	16	8	30	лимит	263	не
instanca_30_v1_2	30	24	12	30	лимит	375	не
instanca_30_v1_3	40	32	16	30	лимит	508	не
instanca_30_v1_4	50	40	20	30	лимит	727	не
instanca_30_v2_0	10	8	4	30	33.73341	47	оптимально
instanca_30_v2_1	20	16	8	30	лимит	232	не
instanca_30_v2_2	30	24	12	30	лимит	332	не
instanca_30_v2_3	40	32	16	30	лимит	566	не
instanca_30_v2_4	50	40	20	30	лимит	734	не
instanca_30_v3_0	10	8	4	30	120.99505	58	оптимально
instanca_30	20	16	8	30	лимит	265	не

_v3_1							
instanca_30 _v3_2	30	24	12	30	лимит	332	не
instanca_30 _v3_3	40	32	16	30	лимит	468	не
instanca_30 _v3_4	50	40	20	30	лимит	634	не

Табела 3 – yices резултати

инстанца	**n**	**m**	**k**	**граница цена**	**време извршавања**	**функција циља**	**оптималност**
instanca_10 _v1_0	10	8	4	10	0.27270	19	оптимално
instanca_10 _v1_1	20	16	8	10	1.37987	13	оптимално
instanca_10 _v1_2	30	24	12	10	9.35122	9	оптимално
instanca_10 _v1_3	40	32	16	10	102.90263	10	оптимално
instanca_10 _v1_4	50	40	20	10	63.00013	6	оптимално
instanca_10 _v2_0	10	8	4	10	0.80062	32	оптимално
instanca_10 _v2_1	20	16	8	10	2.06623	15	оптимално
instanca_10 _v2_2	30	24	12	10	5.77643	12	оптимално
instanca_10 _v2_3	40	32	16	10	лимит	47	не
instanca_10 _v2_4	50	40	20	10	58.69033	5	оптимално
instanca_10 _v3_0	10	8	4	10	0.47500	28	оптимално
instanca_10 _v3_1	20	16	8	10	2.67273	15	оптимално
instanca_10 _v3_2	30	24	12	10	5.74300	14	оптимално
instanca_10	40	32	16	10	127.04228	22	оптимално

_v3_3							
instanca_10_v3_4	50	40	20	10	45.17415	4	оптимально
instanca_20_v1_0	10	8	4	20	0.62317	64	оптимально
instanca_20_v1_1	20	16	8	20	4.18751	44	оптимально
instanca_20_v1_2	30	24	12	20	10.23747	32	оптимально
instanca_20_v1_3	40	32	16	20	65.30085	39	оптимально
instanca_20_v1_4	50	40	20	20	134.62382	37	оптимально
instanca_20_v2_0	10	8	4	20	0.34113	45	оптимально
instanca_20_v2_1	20	16	8	20	11.65153	55	оптимально
instanca_20_v2_2	30	24	12	20	15.40874	41	оптимально
instanca_20_v2_3	40	32	16	20	83.95440	50	оптимально
instanca_20_v2_4	50	40	20	20	244.47584	42	оптимально
instanca_20_v3_0	10	8	4	20	1.06771	48	оптимально
instanca_20_v3_1	20	16	8	20	4.42195	46	оптимально
instanca_20_v3_2	30	24	12	20	122.42909	56	оптимально
instanca_20_v3_3	40	32	16	20	58.96768	52	оптимально
instanca_20_v3_4	50	40	20	20	239.50304	42	оптимально
instanca_30_v1_0	10	8	4	30	0.44853	61	оптимально
instanca_30_v1_1	20	16	8	30	лимит	122	не
instanca_30	30	24	12	30	32.77260	46	оптимально

_v1_2							
instanca_30_v1_3	40	32	16	30	112.92803	50	оптимално
instanca_30_v1_4	50	40	20	30	293.46493	74	оптимално
instanca_30_v2_0	10	8	4	30	0.46551	47	оптимално
instanca_30_v2_1	20	16	8	30	4.20332	64	оптимално
instanca_30_v2_2	30	24	12	30	81.46584	80	оптимално
instanca_30_v2_3	40	32	16	30	77.51944	59	оптимално
instanca_30_v2_4	50	40	20	30	478.72400	88	оптимално
instanca_30_v3_0	10	8	4	30	0.74204	58	оптимално
instanca_30_v3_1	20	16	8	30	7.24936	71	оптимално
instanca_30_v3_2	30	24	12	30	21.56279	72	оптимално
instanca_30_v3_3	40	32	16	30	лимит	102	не
instanca_30_v3_4	50	40	20	30	428.88693	71	оптимално

Табела 4 – sat4j резултати

инстанца	**n**	**m**	**k**	**граница цена**	**време извршавања**	**функција циља**	**оптималност**
instanca_10_v1_0	10	8	4	10	0.84628	19	оптимално
instanca_10_v1_1	20	16	8	10	1.85826	13	оптимално
instanca_10_v1_2	30	24	12	10	1.88663	9	оптимално
instanca_1	40	32	16	10	3.99021	10	оптимално

0_v1_3							
instanca_10_v1_4	50	40	20	10	2.87292	6	оптимално
instanca_10_v2_0	10	8	4	10	1.59641	32	оптимално
instanca_10_v2_1	20	16	8	10	4.22341	15	оптимално
instanca_10_v2_2	30	24	12	10	23.07715	12	оптимално
instanca_10_v2_3	40	32	16	10	468.67919	17	оптимално
instanca_10_v2_4	50	40	20	10	2.47125	5	оптимално
instanca_10_v3_0	10	8	4	10	0.94035	18	оптимално
instanca_10_v3_1	20	16	8	10	7.84582	15	оптимално
instanca_10_v3_2	30	24	12	10	9.22960	14	оптимално
instanca_10_v3_3	40	32	16	10	лимит	29	не
instanca_10_v3_4	50	40	20	10	1.28272	4	оптимално
instanca_20_v1_0	10	8	4	20	1.45680	64	оптимално
instanca_20_v1_1	20	16	8	20	43.36932	44	оптимално
instanca_20_v1_2	30	24	12	20	лимит	35	не
instanca_20_v1_3	40	32	16	20	лимит	53	не
instanca_20_v1_4	50	40	20	20	лимит	49	не
instanca_20_v2_0	10	8	4	20	1.26532	45	оптимално
instanca_20_v2_1	20	16	8	20	41.44681	55	оптимално
instanca_2	30	24	12	20	лимит	42	не

0_v2_2							
instanca_20_v2_3	40	32	16	20	лимит	60	не
instanca_20_v2_4	50	40	20	20	лимит	55	не
instanca_20_v3_0	10	8	4	20	0.84878	48	оптимально
instanca_20_v3_1	20	16	8	20	154.21317	46	оптимально
instanca_20_v3_2	30	24	12	20	лимит	72	не
instanca_20_v3_3	40	32	16	20	лимит	58	не
instanca_20_v3_4	50	40	20	20	лимит	46	не
instanca_30_v1_0	10	8	4	30	0.99418	61	оптимально
instanca_30_v1_1	20	16	8	30	213.31572	85	оптимально
instanca_30_v1_2	30	24	12	30	197.65570	46	оптимально
instanca_30_v1_3	40	32	16	30	лимит	58	не
instanca_30_v1_4	50	40	20	30	лимит	97	не
instanca_30_v2_0	10	8	4	30	0.70453	47	оптимально
instanca_30_v2_1	20	16	8	30	270.30378	64	оптимально
instanca_30_v2_2	30	24	12	30	лимит	86	не
instanca_30_v2_3	40	32	16	30	лимит	89	не
instanca_30_v2_4	50	40	20	30	лимит	105	не
instanca_30_v3_0	10	8	4	30	0.99151	58	оптимально
instanca_3	20	16	8	30	82.06944	71	оптимально

0_v3_1							
instanca_30_v3_2	30	24	12	30	лимит	81	не
instanca_30_v3_3	40	32	16	30	лимит	114	не
instanca_30_v3_4	50	40	20	30	лимит	81	не

У наредним тебелама које је односе на резултате извршавања хеуристичких метода дати су најбоље решење које је хеуристичка метода дала после одређеног броја позива, просечно време извршавања, просечно процентуално одступање, стандардна девијација, а за популационе хеуристике и средњи број генерација до достизања птималног решења. Ови подаци имају за циљ да измере колика је стабилност хеуристичких метода.

Свака хеуристичка метода се позива одређен број и при сваком позиву може дати различита решења. Ако је метода позвана n пута и у i-том извршавању је дала решење sol_i , извршавала се t_i , и у случају хеуристичких метода имала је g_i генерација, ови подаци се рачунају на следећи начин:

- најбоље решење је решење са најмањом вредношћу функције циља, и нека је та вредност F
- просечно време извршавања по формули: $\sum_{i=1}^{n} t_i$
- просечно процентуално одступање по формули: $agap = \frac{1}{n}\sum_{i=1}^{n} gap_i$, где је

$$gap_i = 100 * \frac{|sol_i - F|}{|F|}$$

- стандардна девијација се рачуна по формули: $\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (gap\,i - agap)^2}$
- средњи број генерација по формули: $agen = \frac{1}{n}\sum_{i=1}^{n} g_i$

Табела 5 – генетски алгоритам резултати

инстанца	**n**	**m**	**k**	**граница цена**	**просечно време извршавања**	**функција циља**	**просечно процентуално одсупање**	**стандардна девијација**	**средње итерација**
instanca_10_v1_0	10	8	4	10	0.03500	19	0.52631	1.57894	22.00
instanca_10_v1_1	20	16	8	10	0.07400	13	3.07692	3.76844	26.90
instanca_10_v1_2	30	24	12	10	0.16400	9	31.11111	17.77777	30.70
instanca_10_v1_3	40	32	16	10	0.19200	10	22.00000	17.77638	32.20
instanca_10_v1_4	50	40	20	10	0.24200	6	36.66666	24.49489	31.70
instanca_10_v2_0	10	8	4	10	0.03900	32	0	0	22.40
instanca_10_v2_1	20	16	8	10	0.08700	15	4.00000	3.26598	28.40
instanca_10_v2_2	30	24	12	10	0.14400	12	6.66666	8.16496	28.30
instanca_10_v2_3	40	32	16	10	0.18500	18	26.66666	16.62958	30.60
instanca_10_v2_4	50	40	20	10	0.29600	7	30.00000	22.54247	34.00
instanca_10_v3_0	10	8	4	10	0.04000	28	3.57142	3.57142	22.20
instanca_10_v3_1	20	16	8	10	0.07000	15	0	0	25.70
instanca_10_v3_2	30	24	12	10	0.15000	15	17.33333	9.97775	30.50
instanca_10_v3_3	40	32	16	10	0.20000	22	9.09090	7.04178	30.50
instanca_10_v3_4	50	40	20	10	0.26300	5	36.00000	37.73592	32.80
instanca_20_v1_0	10	8	4	20	0.04300	64	1.40625	4.21875	25.00
instanca_20	20	16	8	20	0.09300	44	6.81818	5.47345	31.40

_v1_1									
instanca_20_v1_2	30	24	12	20	0.17000	33	3.63636	2.96907	32.80
instanca_20_v1_3	40	32	16	20	0.21600	39	31.28205	14.67598	35.60
instanca_20_v1_4	50	40	20	20	0.31400	38	7.89473	4.07682	34.90
instanca_20_v2_0	10	8	4	20	0.03600	45	0	0	22.60
instanca_20_v2_1	20	16	8	20	0.07100	55	1.81818	2.69679	28.90
instanca_20_v2_2	30	24	12	20	0.17300	41	5.36585	4.33570	32.90
instanca_20_v2_3	40	32	16	20	0.20600	51	9.41176	6.83748	43.90
instanca_20_v2_4	50	40	20	20	0.30800	46	9.13043	8.06401	37.30
instanca_20_v3_0	10	8	4	20	0.03600	48	0	0	21.80
instanca_20_v3_1	20	16	8	20	0.08300	46	5.86956	4.56521	30.60
instanca_20_v3_2	30	24	12	20	0.16600	59	0.50847	1.52542	32.20
instanca_20_v3_3	40	32	16	20	0.22100	55	6.36363	4.00413	34.30
instanca_20_v3_4	50	40	20	20	0.30600	45	12.66666	9.11111	38.10
instanca_30_v1_0	10	8	4	30	0.03800	61	0.16393	0.49180	23.00
instanca_30_v1_1	20	16	8	30	0.07100	85	3.76470	5.28234	27.30
instanca_30_v1_2	30	24	12	30	0.16800	46	4.34782	4.95728	33.70
instanca_30_v1_3	40	32	16	30	0.23800	53	9.81132	6.73983	36.90
instanca_30_v1_4	50	40	20	30	0.29800	80	8.25000	6.91917	35.50
instanca_30	10	8	4	30	0.04500	47	2.76595	3.43733	24.50

_v2_0									
instanca_30_v2_1	20	16	8	30	0.07200	64	1.56250	1.71163	27.70
instanca_30_v2_2	30	24	12	30	0.14300	81	6.66666	3.71192	30.00
instanca_30_v2_3	40	32	16	30	0.23900	62	20.16129	13.99141	36.50
instanca_30_v2_4	50	40	20	30	0.30800	94	8.29787	8.16866	39.10
instanca_30_v3_0	10	8	4	30	0.04200	58	0	0	24.90
instanca_30_v3_1	20	16	8	30	0.09000	76	9.47368	3.61782	29.10
instanca_30_v3_2	30	24	12	30	0.19200	72	5.00000	2.07869	35.90
instanca_30_v3_3	40	32	16	30	0.18900	107	5.23364	4.07802	33.80
instanca_30_v3_4	50	40	20	30	0.30700	72	11.25000	4.49837	35.50

Табела 6 – локална претрага резултати

инстанца	**n**	**m**	**k**	**граница цена**	**време извршавања**	**функција циља**	**просечно процентуално одступање**	**стандардна девијација**
instanca_10_v1_0	10	8	4	10	0.00000	19	3.15789	6.31578
instanca_10_v1_1	20	16	8	10	0.00000	13	18.46153	15.46134
instanca_10_v1_2	30	24	12	10	0.00000	13	26.15384	19.52088
instanca_10_v1_3	40	32	16	10	0.00150	19	33.15789	22.33589
instanca_10_v1_4	50	40	20	10	0.00150	14	47.14285	31.97575
instanca_10_v2_0	10	8	4	10	0.00000	32	4.68750	6.73870

instanca_10_v2_1	20	16	8	10	0.00000	15	20.66666	19.19490
instanca_10_v2_2	30	24	12	10	0.00150	12	45.83333	33.17671
instanca_10_v2_3	40	32	16	10	0.00150	24	48.33333	20.08661
instanca_10_v2_4	50	40	20	10	0.00150	17	34.70588	28.75141
instanca_10_v3_0	10	8	4	10	0.00000	28	6.78571	2.50000
instanca_10_v3_1	20	16	8	10	0.00000	16	13.75000	9.60143
instanca_10_v3_2	30	24	12	10	0.00150	22	23.18181	25.58812
instanca_10_v3_3	40	32	16	10	0.00150	28	18.57142	12.35115
instanca_10_v3_4	50	40	20	10	0.00150	13	43.84615	33.53881
instanca_20_v1_0	10	8	4	20	0.00000	64	21.09375	7.23658
instanca_20_v1_1	20	16	8	20	0.00000	48	26.04166	15.76327
instanca_20_v1_2	30	24	12	20	0.00150	38	26.31578	13.87518
instanca_20_v1_3	40	32	16	20	0.00000	48	36.66666	23.55549
instanca_20_v1_4	50	40	20	20	0.00150	51	24.11764	16.12260
instanca_20_v2_0	10	8	4	20	0.00150	45	1.77777	3.55555
instanca_20_v2_1	20	16	8	20	0.00150	57	17.54386	13.83631
instanca_20_v2_2	30	24	12	20	0.00150	45	26.00000	18.16386
instanca_20_v2_3	40	32	16	20	0.00150	66	13.48484	6.55029
instanca_20_v2_4	50	40	20	20	0.00160	54	41.29629	19.08485

instanca_20_v3_0	10	8	4	20	0.00000	48	4.16666	6.45497
instanca_20_v3_1	20	16	8	20	0.00000	46	25.43478	26.08786
instanca_20_v3_2	30	24	12	20	0.00150	59	14.74576	11.01173
instanca_20_v3_3	40	32	16	20	0.00150	62	16.93548	14.30400
instanca_20_v3_4	50	40	20	20	0.00150	61	18.85245	8.98653
instanca_30_v1_0	10	8	4	30	0.00000	61	11.31147	21.08900
instanca_30_v1_1	20	16	8	30	0.00150	90	14.00000	7.92246
instanca_30_v1_2	30	24	12	30	0.00000	47	66.59574	34.16268
instanca_30_v1_3	40	32	16	30	0.00000	65	41.69230	30.69568
instanca_30_v1_4	50	40	20	30	0.00160	101	21.28712	10.93824
instanca_30_v2_0	10	8	4	30	0.00000	47	18.72340	14.13893
instanca_30_v2_1	20	16	8	30	0.00150	64	17.96875	18.10411
instanca_30_v2_2	30	24	12	30	0.00000	87	22.29885	12.05746
instanca_30_v2_3	40	32	16	30	0.00150	94	14.46808	8.57950
instanca_30_v2_4	50	40	20	30	0.00150	126	8.88888	7.34116
instanca_30_v3_0	10	8	4	30	0.00000	58	9.48275	8.96717
instanca_30_v3_1	20	16	8	30	0.00000	93	9.89247	11.46863
instanca_30_v3_2	30	24	12	30	0.00000	75	13.60000	11.13632
instanca_30_v3_3	40	32	16	30	0.00000	121	12.89256	10.00102

instanca_30_v3_4	50	40	20	30	0.00150	89	18.76404	11.42592

Табела 7 – резултати хибридизације локалне претраге и генетског алгоритма

инстанца	**n**	**m**	**k**	**граница цена**	**време извршавања**	**функција циља**	**просечно процентуално одсупање**	**стандардна девијација**	**средње итерација**
instanca_10_v1_0	10	8	4	10	0.04800	19	0.52631	1.57894	22.80
instanca_10_v1_1	20	16	8	10	0.10700	13	0	0	25.40
instanca_10_v1_2	30	24	12	10	0.20500	9	23.33333	15.27525	29.70
instanca_10_v1_3	40	32	16	10	0.27100	10	10.00000	15.49193	32.20
instanca_10_v1_4	50	40	20	10	0.40700	6	23.33333	20.00000	32.30
instanca_10_v2_0	10	8	4	10	0.05200	32	0	0	23.00
instanca_10_v2_1	20	16	8	10	0.13000	15	1.33333	2.66666	27.80
instanca_10_v2_2	30	24	12	10	0.18400	12	4.16666	8.53912	29.00
instanca_10_v2_3	40	32	16	10	0.23600	18	21.11111	14.22916	30.60
instanca_10_v2_4	50	40	20	10	0.41600	7	10.00000	14.35696	34.80
instanca_10_v3_0	10	8	4	10	0.05200	28	1.42857	2.85714	22.50
instanca_10_v3_1	20	16	8	10	0.10700	15	0	0	25.80
instanca_10_v3_2	30	24	12	10	0.21100	14	11.42857	10.20204	30.70
instanca_10_v3_3	40	32	16	10	0.28000	22	4.090909	4.28817	30.60
instanca_10	50	40	20	10	0.41300	4	25.00000	35.35533	33.70

_v3_4									
instanca_20_v1_0	10	8	4	20	0.05200	64	0	0	24.20
instanca_20_v1_1	20	16	8	20	0.13600	44	5.00000	5.45454	29.70
instanca_20_v1_2	30	24	12	20	0.21700	32	1.56250	2.88110	32.60
instanca_20_v1_3	40	32	16	20	0.31600	40	14.00000	14.88287	36.60
instanca_20_v1_4	50	40	20	20	0.44300	37	7.56756	4.32432	36.00
instanca_20_v2_0	10	8	4	20	0.05100	45	0	0	23.20
instanca_20_v2_1	20	16	8	20	0.11200	55	0	0	28.70
instanca_20_v2_2	30	24	12	20	0.21400	41	2.43902	3.27229	32.40
instanca_20_v2_3	40	32	16	20	0.29000	50	7.80000	5.96322	34.50
instanca_20_v2_4	50	40	20	20	0.41100	43	6.51162	4.26286	37.90
instanca_20_v3_0	10	8	4	20	0.04800	48	0	0	22.40
instanca_20_v3_1	20	16	8	20	0.12300	46	3.04347	3.39576	29.30
instanca_20_v3_2	30	24	12	20	0.18200	59	0.67796	2.03389	28.80
instanca_20_v3_3	40	32	16	20	0.29000	54	4.44444	4.77188	33.80
instanca_20_v3_4	50	40	20	20	0.41800	44	6.81818	4.87445	38.70
instanca_30_v1_0	10	8	4	30	0.05000	61	0	0	22.70
instanca_30_v1_1	20	16	8	30	0.10900	85	1.41176	3.52156	27.90
instanca_30_v1_2	30	24	12	30	0.22100	46	0	0	33.70
instanca_30	40	32	16	30	0.33200	52	6.73076	3.67403	36.80

_v1_3									
instanca_30_v1_4	50	40	20	30	0.43800	79	3.29113	5.83378	40.40
instanca_30_v2_0	10	8	4	30	0.06000	47	1.06383	3.19148	24.60
instanca_30_v2_1	20	16	8	30	0.11400	64	1.40625	1.77465	27.80
instanca_30_v2_2	30	24	12	30	0.20300	81	4.93827	5.09025	31.90
instanca_30_v2_3	40	32	16	30	0.32100	60	12.50000	11.11180	37.20
instanca_30_v2_4	50	40	20	30	0.41700	88	6.59090	5.15262	38.30
instanca_30_v3_0	10	8	4	30	0.05400	58	0	0	23.70
instanca_30_v3_1	20	16	8	30	0.13100	71	7.32394	8.42248	28.10
instanca_30_v3_2	30	24	12	30	0.22000	72	4.30555	4.23326	34.80
instanca_30_v3_3	40	32	16	30	0.28300	100	11.50000	5.38980	34.30
instanca_30_v3_4	50	40	20	30	0.41800	71	3.94366	3.49568	37.00

Литература

[1] Predrag Janičić, Mladen Nikolić. 2010. Veštačka inteligencija. Elektronsko izdanje

[2] Wen Wan and Jeffrey B. Birch. An improved Hybrid Genetic Algorithm With a New Local Search Procedure. Journal of Applied Mathematics Volume 2013 (2013), Article ID 103591

[3] CPLEX упутство, ftp://public.dhe.ibm.com/software/websphere/ilog/docs/optimization/cplex/ps_usrmancplex.pdf, приступљено октобра 2014.

[4] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. 2006. Solving SAT and SAT Modulo Theories: From an abstract Davis--Putnam--Logemann--Loveland procedure to DPLL(*T*). *J. ACM* 53, 6 (November 2006), 937-977.

[5] yices упутство, http://yices.csl.sri.com/papers/manual.pdf, приступљено октобра 2014.

[6] Bruno Dutertre and Leonardo De Moura. Integrating simplex with DPLL (T). *Computer Science Laboratory, SRI International, Tech. Rep. SRI-CSL-06-01* (2006).

[7] Stefan Misković, Zorica Stanimirović. A Memetic Algorithm for Solving Two Variants of the Two-stage Uncapaciteted Facility Location Problem. Information technology and control, 2013, Vol. 42, No. 2

[8] Stuart J. Russell and Peter Norvig. 2003. *Artificial Intelligence: A Modern Approach* (2 ed.). Pearson Education

[9] http://www.cril.univ-artois.fr/PB12/format.pdf

[10] minion документација, http://minion.sourceforge.net/manual.html, приступљено октобра 2014.

[11] G.S. Tseitin: On the complexity of derivation in propositional calculus. In: Slisenko, A.O. (ed.) Structures in Constructive Mathematics and Mathematical Logic, Part II, Seminars in Mathematics (translated from Russian), pp. 115–125. Steklov Mathematical Institute (1968)

[12] Moscato, P. (1989). "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms". Caltech Concurrent Computation Program (report 826).

[13] Holland, John H. "Genetic algorithms." Scientific american 267, no. 1 (1992): 66-72.

[14] Intriligator, Michael D.Mathematical optimization and economic theory. Vol. 39. Siam, 2002.
[15] Kantorovich, Leonid V. "Mathematical methods of organizing and planning production." Management Science 6.4 (1960): 366-422.
[16] O.D. Sirotenko, A.V. Romanenkov Mathematical Models of Life Support Systems – Vol II – Mathematical Models of Agricultural Supply