UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Mirko S. Stojadinović

REŠAVANJE PROBLEMA CSP TEHNIKAMA SVOĐENJA NA PROBLEM SAT

doktorska disertacija

Beograd, 2016.

UNIVERSITY OF BELGRADE
FACULTY OF MATHEMATICS

Mirko S. Stojadinović

SOLVING OF CONSTRAINT SATISFACTION PROBLEMS BY REDUCTION TO SAT

Doctoral Dissertation

Belgrade, 2016.

Mentor:

dr Filip Marić, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Predrag Janičić, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Nenad Mitić, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Mladen Nikolić, docent
Univerzitet u Beogradu, Matematički fakultet

dr Mirko Vujošević, redovni profesor
Univerzitet u Beogradu, Fakultet organizacionih nauka

Datum odbrane: _____

Zahvalnost

Želeo bih ovom prilikom da se zahvalim svom mentoru dr Filipu Mariću na ogromnom trudu i vremenu koje je uložio u naš zajednički rad. Profesorov entuzijazam je u mnogim trenucima bio veoma "zarazan" i uticao je da imam jaku motivaciju za rad na doktoratu. Profesor mi je u velikoj meri olakšao i učinio lepši rad na doktoratu svojom dobronamernošću i nesebičnošću, što su kvaliteti koje svaki doktorand kod svog mentora samo poželeti može. Takođe, hteo bih da se zahvalim profesoru dr Predragu Janičiću na vođenju grupe za automatsko rezonovanje i projekta Ministarstva, u čijim sam aktivnostima učestvovao, kao i na velikom broju korisnih saveta vezanih za bavljenje naukom u Srbiji, ali sa ciljem da se naprave svetski doprinosi. Profesor mi je veliki broj puta dao korisne komentare, kako na izlaganjima tako i prilikom čitanja rukopisa čiji sam autor bio. Veliku zahvalnost dugujem i dr Mladenu Nikoliću, sa kojim sam puno puta diskutovao o tehnikama mašinskog učenja i koji mi je u značajnoj meri pomogao da unapredim svoje znanje u ovoj oblasti. Njemu, kao i profesorima dr Nenadu Mitiću i dr Mirku Vujoševiću, dugujem zahvalnost jer su pristali da budu članovi komisije za pregled i ocenu ove teze. Svi do sada pomenuti su u značajnoj meri doprineli poboljšanju kvaliteta ovog teksta, dajući mnoge korisne primedbe i sugestije. Zahvalio bih se i svim profesorima i kolegama sa Katedre za računarstvo i informatiku, jer su mi držanjem kurseva, diskusijama i korisnim savetima pomogli tokom celog mog studiranja i rada na Matematičkom fakultetu. Među njima, istakao bih pomoć koleginice Sane Stojanović Đurđević koja mi je svojim dobronamernim savetima i komentarima puno pomogla na doktorskim studijama, kao i kolege Milana Bankovića, koji je više puta čitao radove čiji sam autor bio i dao puno korisnih komentara za poboljšanje kvaliteta tih radova.

Naslov disertacije: Rešavanje problema CSP tehnikama svođenja na problem SAT

Rezime: Mnogi realni problemi se danas mogu predstaviti u obliku problema zadovoljenja ograničenja (CSP) i zatim rešiti nekom od mnogobrojnih tehnika za rešavanje ovog problema. Jedna od tehnika podrazumeva svođenje na problem SAT, tj. problem iskazne zadovoljivosti. Promenljive i ograničenja problema CSP se prevode (kodiraju) u SAT instancu, ona se potom rešava pomoću modernih SAT rešavača i rešenje se, ako postoji, prevodi u rešenje problema CSP. Glavni cilj ove teze je unapređenje rešavanja problema CSP svođenjem na SAT.

Razvijena su dva nova hibridna kodiranja (prevođenja u SAT formulu) koja kombinuju dobre strane postojećih kodiranja. Dat je dokaz korektnosti jednog od kodiranja koji do sada nije postojao u literaturi. Razvijen je sistem *meSAT* koji omogućava svođenje problema CSP na SAT pomoću četiri osnovna i dva hibridna kodiranja, kao i rešavanje problema CSP svođenjem na dva problema srodna problemu SAT, SMT i PB.

Razvijen je portfolio za automatski odabir kodiranja/rešavača za ulaznu instancu koju je potrebno rešiti i pokazano je da je razvijeni portfolio uporediv sa najefikasnijim savremenim pristupima. Prikazan je novi pristup zasnovan na kratkim vremenskim ograničenjima sa ciljem da se značajno smanji vreme pripreme portfolija. Pokazano je da se ovim pristupom dobijaju rezultati konkurentni onima koji se dobijaju korišćenjem standardnog vremena za pripremu. Izvršeno je poređenje nekoliko tehnika mašinskog učenja, sa ciljem da se ustanovi koja od njih je pogodnija za pristup sa kratkim treniranjem.

Prikazan je jedan realan problem, problem raspoređivanja kontrolora leta, kao i tri njegova modela. Veliki broj metoda rešavanja i raznovrsnih rešavača je upotrebljeno za rešavanje ovog problema. Razvijeno je više optimizacionih tehnika koje imaju za cilj pronalaženje optimalnih rešenja problema. Pokazuje se da je najefikasnija hibridna tehnika koja kombinuje svođenje na SAT i lokalnu pretragu.

Razmotren je i problem sudoku, kao i postojeće tehnike rešavanja sudoku zagonetki većih dimenzija od 9×9. Pokazuje se da je u rešavanju ovih zagonetki najefikasnije već postojeće svođenje na SAT. Unapređen je postojeći algoritam za generisanje velikih sudoku zagonetki. Pokazano je da jednostavna pravila preprocesiranja dodatno unapređuju brzinu generisanja sudokua.

Ključne reči: CSP, SAT kodiranje, portfolio, kontrolor, raspored, sudoku

Naučna oblast: računarstvo

Uža naučna oblast: automatsko rezonovanje

UDK broj: 004.832.3(043.3)

Dissertation title: Solving of Constraint Satisfaction Problems by reduction to SAT

Abstract: Many real-world problems can be modeled as constraint satisfaction problems (CSPs) and then solved by one of many available techniques for solving these problems. One of the techniques is reduction to SAT, i.e. Boolean Satisfiability Problem. Variables and constraints of CSP are translated (encoded) to SAT instance, that is then solved by state-of-the-art SAT solvers and solution, if exists, is translated to the solution of the original CSP. The main aim of this thesis is to improve CSP solving techniques that are using reduction to SAT.

Two new hybrid encodings of CSPs to SAT are presented and they combine good sides of the existing encodings. We give the proof of correctness of one encoding that did not exist in literature. We developed system *meSAT* that enables reduction of CSPs to SAT by using 4 basic and 2 hybrid encodings. The system also enables solving of CSPs by reduction to two problems related to SAT, SMT and PB.

We developed a portfolio for automated selection of encoding/solver to be used on some new instance that needs to be solved. The developed portfolio is comparable with the state-of-the-art portfolios. We developed a hybrid approach based on short solving timeouts with the aim of significantly reducing the preparation time of a portfolio. By using this approach, we got results comparable to the ones obtained by using preparation time of usual length. We made comparison between several machine learning techniques with the aim to find out which one is the best suited for the short training approach.

The problem of assigning air traffic controllers to shifts is described and three models of this problem are presented. We used a large number of different solving methods and a diverse set of solvers for solving this problem. We developed optimization techniques that aim to find optimal solutions of the problem. A hybrid technique combining reduction to SAT and local search is shown to be the most efficient one.

We also considered sudoku puzzles and the existing techniques of solving the puzzles of greater size than 9×9. Amongst the used techniques, the existing reduction to SAT is the most efficient in solving these puzzles. We improved the existing algorithm for generating large sudoku puzzles. It is shown that simple preprocessing rules additionally improve speed of generating large sudokus.

Keywords: CSP, SAT encoding, portfolio, air traffic controler, schedule, sudoku

Research area: computer science

Research sub-area: automated reasoning

UDC number: 004.832.3(043.3)

Sadržaj

Slike

Tabele

Glava 1

Uvod

1.1 Osnovni pojmovi

Računari se od svog nastanka upotrebljavaju u rešavanju velikog broja realnih problema. Poslednjih decenija, omogućeno je rešavanje sve težih i kompleksnijih problema, a razlozi za uspešnost primene računara su mnogobrojni. Jedan od najbitnijih je upotreba sve boljeg i sofisticiranijeg softvera – razvijen je veliki broj pristupa za rešavanje realnih problema. Očekivano, ispostavlja se da su za različite vrste problema pogodni različiti pristupi, kao i da računarski programi specijalno razvijeni za neku užu klasu problema skoro po pravilu nadmašuju programe opštije namene, razvijene za rešavanje šire klase problema. Intenzivan razvoj sve boljih i boljih algoritama prethodnih decenija takođe je u ogromnoj meri uticao na efikasno rešavanja problema pomoću računara. Na uspešnost primene računara veliki uticaj ima i izuzetan napredak u performansama računara čime je omogućeno da budu rešeni problemi za koje je pre samo par desetina godina bilo nezamislivo da mogu biti rešeni pomoću računara.

Jedan od čestih načina rešavanja velikog broja problema (npr. problema raspoređivanja i planiranja) je korišćenjem programske paradigme koja se zove *programiranje ograničenja* (eng. *Constraint programming*), skraćeno CP [112]. U ovoj paradigmi se odnosi između promenljivih predstavljaju ograničenjima – ograničenje na skupu promenljivih određuje koje kombinacije vrednosti iz domena promenljivih su dopuštene, a koje nisu. Za razliku od imperativnog programiranja kod koga se zadaje niz koraka koje treba izvršiti, kod CP pristupa zadaju se svojstva rešenja koje je potrebno pronaći, a sistem primenom određenih algoritama pronalazi takva rešenja. Otuda se CP svrstava u skup programskih paradigmi koji se zove deklarativno

programiranje.

Problem zadovoljenja ograničenja (eng. *Constraint Satisfaction Problem*), skraćeno CSP, je matematički problem definisan na konačnom skupu promenljivih, čije su vrednosti iz zadatih konačnih domena (najčešće podskupova skupa celih brojeva), pri čemu se zadaje konačan skup ograničenja koja ove promenljive moraju da zadovoljavaju. *Problem optimizacije uz ograničenja* ili *problem uslovne optimizacije* (eng. *Constraint Optimization Problem*), skraćeno COP, je problem koji se definiše na sličan način kao problem CSP, ali je razlika u tome što definicija problema COP obuhvata i funkciju cilja. Pri rešavanju problema COP, cilj je naći rešenje koje zadovoljava sva ograničenja i maksimizuje/minimizuje funkciju cilja. Rešavanja problema pomoću CP pristupa sastoji se iz dva dela:

1. *Modeliranje* problema kao problema CSP/COP[1]. Potrebno je odrediti koje promenljive će se koristiti, koje među njima će biti deo rešenja a koje će biti samo pomoćne. Potrebno je zadati i ograničenja među promenljivama i rešenje problema mora da zadovoljava sva zadata ograničenja. Sva ograničenja je potrebno formulisati u obliku koji je dopušten jezikom za modeliranje koji se koristi.

2. *Rešavanje* problema CSP (u ovog tekstu se razmatra samo rešavanje pomoću računara, a ne ručno rešavanje). U zavisnosti od modela dobijenog u fazi modeliranja, može se koristiti neka od odgovarajućih (pogodnih) metoda i rešavača za rešavanje problema. Na primer, ukoliko je model problema takav da sadrži promenljive sa celobrojnim domenima, verovatno je da neće moći da se primene metode razvijene za iskazne promenljive. Problem se može rešiti korišćenjem metoda specijalizovanih za taj problem (ili za sličnu vrstu problema), ili upotrebom opštih CSP rešavača, programa razvijenih da rešavaju probleme CSP u nekom opštem obliku i u konkretnoj sintaksi.

Problemi CSP su postojali i rešavani su od davnina. Na primer, raspoređivanje radnika po smenama ili zadacima je problem koji se često tokom istorije javljao. Tezejevo kretanje kroz lavirint na Kritu se spominje kao primer korišćenja *pretrage sa povratkom* (eng. *backtracking*), a ovo je danas centralna metoda za rešavanje problema CSP. Problem *8 kraljica* koji je jedan od najpoznatijih problema CSP, nastao je sredinom devetnaestog veka.

[1]Nadalje ćemo korisiti samo termin CSP, ali se uvek misli bilo na problem CSP ili COP

Moderna era u rešavanju problema CSP počinje sredinom 1960-ih. Postojala su dva glavna i tokom dugog perioda vrlo odvojena pravca razvoja u oblasti CP. Prvi se odnosio na razvoj reprezentacije, odnosno razvoj velikog broja jezika koji su se koristili za modeliranje problema. Drugi se odnosio na unapređenje algoritama za rešavanje problema CSP, često sa naglaskom na teorijskom doprinosu[2]. U tom pravcu, razmatrano je rešavanje jednostavnog ali dovoljno opšteg oblika problema CSP, zajedno sa nekim varijantama tog problema[3]. Tokom devedesetih godina dvadesetog veka dolazi do laganog spajanja ova dva pravca i stvaranja jedinstvene zajednice CP.

Od početka dvadesetprvog veka počinju redovno da se održavaju takmičenja koja imaju za cilj da ohrabre učesnike da unaprede postojeće i razviju nove tehnike koje koriste CSP rešavači. Najpoznatije takmičenje je *MiniZinc Challenge*. Radovi iz oblasti CP predstavljaju se na mnogim konferencijama iz oblasti veštačke inteligencije, a postoje i specijalizovane konferencije posvećene oblasti CP. Najprestižnija je *International Conference on Principles and Practice of Constraint Programming* (CP). Prepoznat je i veliki značaj korišćenja tehnika veštačke inteligencije i operacionih istraživanja u oblasti CP, pa postoji konferencija posebno namenjena ovoj temi: *International Conference on Integration of AI and OR Techniques in Constraint Programming* (CPAIOR). Najprestižniji časopis posvećen oblasti CP je časopis *Constraints*. Knjiga koja najsveobuhvatnije opisuje koncepte vezane za oblast CP je *Handbook of Constraint Programming* [108].

Postoji više metoda za rešavanje problema CSP. Najčešće CSP rešavači traže rešenje na sistematičan način, tj. garantuju da će rešenje biti pronađeno ako postoji. Obično se ovo postiže naizmeničnom pretragom i primenom *propagiranja ograničenja* (eng. *constraint propagation*). Drugi dosta korišćen način za rešavanje problema CSP je lokalna pretraga – ona je vrlo brza ali se njenom primenom u opštem slučaju ne garantuje pronalaženje rešenja ako postoji. Zadnjih godina se sve više koriste hibridni pristupi, koji koriste i sistematičnu i nesistematičnu pretragu.

Jedna od metoda rešavanja problema CSP je svođenjem na *problem iskazne zadovoljivosti*, SAT problem (eng. *Boolean Satisfiability Problem*). Promenljive i ograničenja problema CSP prevode se u iskazne promenljive i klauze problema SAT

[2]Na primer, razvijen je veliki broj konzistencija problema CSP i algoritama za prevođenje bilo kog problema CSP u ta stanja, ali oni nikada nisu ušli u praktičnu upotrebu.

[3]Jedna od varijanti je *težinski CSP* – nije moguće zadovoljiti sva ograničenja problema, pa su ograničenja podeljena u ona koja moraju biti zadovoljena (*tvrda*), i ona koja su opciona i kojima su pridružene težine (*meka*). Potrebno je naći rešenje koje zadovoljava sva tvrda ograničenja i minimizuje sumu težina nezadovoljenih mekih ograničenja.

(instanca problema SAT je predstavljena kao konjunkcija klauza). Ovi problemi su *ekvizadovoljivi* (eng. *equisatisfiable*), tj. zadovoljivost jednog od njih povlači zadovoljivost drugog, a nezadovoljivost jednog od njih povlači nezadovoljivost drugog. Program koji se zove SAT rešavač se pokreće na dobijenoj iskaznoj formuli, pronalazi rešenje (ako postoji) i potom se to rešenje prevodi u rešenje problema CSP. Prethodnih decenija razvijen je veliki broj javno dostupnih SAT rešavača koji uspevaju da reše instance koje se sastoje od miliona klauza, pa se veliki broj teških industrijskih problema rešava svođenjem na SAT.

Problem CSP se može prevesti u problem SAT pomoću više kodiranja, a svako od kodiranja na različit način prevodi promenljive i ograničenja u iskazne promenljive i klauze. Najpoznatija kodiranja su: *direktno kodiranje* (eng. direct), *kodiranje podrške* (support), *kodiranje uređenja* (order) i *logaritamsko kodiranje* (log). Svako od ovih kodiranja pogodnije je od ostalih za rešavanje određenih problema.

Različite metode rešavanja pogodne su za različite klase problema CSP. Prilikom korišćenja metode svođenja na SAT, različitim problemima CSP pogoduju različita kodiranja. Dodatno, pogodan izbor SAT rešavača koji rešava kodiranu instancu takođe zavisi od karakteristika same instance. U okviru rešavanja problema SAT, a u poslednje vreme i u okviru rešavanja problema CSP, razvijeni su mnogi *portfolio pristupi* koji za cilj imaju odabir rešavača pogodnog za određeni problem. Ovi pristupi su većinom zasnovani na tehnikama mašinskog učenja. Prilikom primene svakog od ovih pristupa, prvo se vrši trening na nekom unapred određenom skupu instanci, a zatim se prilikom rešavanja nekog novog skupa instanci koje su od interesa vrši odabir rešavača na osnovu znanja prikupljenog u fazi treninga. Veliki broj ovih pristupa postoji za SAT rešavače, dok je manji broj njih razvijen za CSP rešavače.

U poslednjih nekoliko decenija CP tehnike su uspešno korišćene za rešavanje velikog broja praktičnih problema. Neke od primena su:

- Raspoređivanje aviona koji sleću na odgovarajuće piste.
- Rutiranje vozila. Ovaj problem se sastoji u određivanju putanja za vozila sa ciljem da se posete svi potrošači uz minimalnu cenu.
- Dizajn, analiza rizika i operaciona kontrola kod raznih distribuiranih mreža (električnih, mreža podataka, itd.).
- Konstrukcija 3D modela proteina.
- Otkrivanje nedostataka u električnim sklopovima.

- Konstrukcija efikasnih parsera.

Posebna pažnja je usmerena na generisanje reprezentativnih CSP i SAT instanci, da bi se napravilo što je moguće preciznije poređenje rešavača i tehnika za ove probleme. Veliki korpusi ovih instanci koriste se na takmičenjima, a obuhvataju kako instance iz praktičnih primena, tako i instance koje se odnose na igre i zagonetke.

1.2 Doprinosi teze

Cilj ove teze je da doprinese boljem razumevanju i efikasnijem rešavanju problema CSP pomoću tehnika svođenja na SAT. Konkretno, doprinosi ovog rada su sledeći:

1. Razvijen je sistem *meSAT* (*multiple encodings to SAT*), karakterističan po tome što podržava različite tehnike kodiranja problema CSP prilikom svođenja na problem SAT. Postoji više sistema koji koriste jedno (ili eventualno dva) kodiranja za svođenje na SAT, što nije dovoljno za praktičnu upotrebljivost na širokom dijapazonu problema i stoga je u praksi potrebno isprobati više različitih sistema u rešavanju nekog problema. Poželjno je imati jedinstven sistem koji omogućava izbor između većeg broja kodiranja i *meSAT* je prvi takav sistem. Još jedan cilj razvoja ovakvog sistema je mogućnost nepristrasne evaluacije različitih tehnika svođenja na SAT.

2. Razvijena su hibridna kodiranja koja koriste dobre strane već postojećih, osnovnih kodiranja. Sistem *meSAT* na taj način podržava 4 osnovna i 2 hibridna kodiranja, a pored toga i svođenje na 2 problema bliska problemu SAT: problem SMT (eng. *Satisfiability Modulo Theories*) i problem PB (eng. *Pseudo-Boolean*).

3. Urađena je teorijska analiza različitih kodiranja. Dat je i dokaz korektnosti kodiranja uređenja koji nedostaje u literaturi. Takođe, na primeru nekih konkretnih problema dato je objašnjenje performansi koje različita kodiranja postižu u kontekstu mehanizama i algoritama rešavanja dobijenog problema SAT.

4. Urađena je eksperimentalna evaluacija na velikom korpusu instanci i sa značajnim brojem rešavača. Izvršeno je poređenje različitih tehnika mašinskog

učenja u odabiru CSP rešavača. Razvijen je, implementiran i testiran portfolio pristup ArgoCSP-kNN zasnovan na algoritmu k-najbližih suseda koji je dao najbolje rezultate među korišćenim tehnikama mašinskog učenja. ArgoCSP-kNN je uporediv sa najboljim postojećim portfolio pristupima i daje dobre rezultate kako u izboru kodiranja razvijenih u okviru sistema *meSAT*, tako i u izboru rešavača. Napravljena je i procena uticaja dužine treniranja na kvalitet odabira rešavača, a pristup ArgoCSP-kNN sa kratkim vremenom treniranja (nekoliko sati ili dana) je konkurentan postojećim pristupima koji koriste značajno duže treniranje (obično nekoliko meseci).

5. Rešavan je jedan praktičan problem, *Problem raspoređivanja kontrolora leta* (eng. *Air Traffic Controller Shift Scheduling Problem*), skraćeno *ATCoSSP*. Ručno raspoređivanje kontrolora leta u smene može predstavljati vrlo težak zadatak jer je potrebno rasporediti kontrolore na odgovarajuće pozicije u toku smena da bi u svakom trenutku na svakoj poziciji bio dovoljan broj kontrolora. Svako od zaposlenih ima i želje u vezi sa rasporedom – potrebno je zadovoljiti što je veći broj želja – pa je ovo vrsta optimizacionog problema. Predstavljena su tri modela problema. Problem je rešavan pomoću velikog broja metoda i rešavača, kao i hibridizacijom svođenja na SAT i lokalne pretrage da bi se raspoređivanje izvršilo što efikasnije. Generisane instance problema *ATCoSSP* korišćene su na zvaničnim takmičenjima rešavača.

6. Razmotren je i problem rešavanja i generisanja velikih sudoku zagonetki (većih od 9×9). Postojeće svođenje na SAT koje do sada nije detaljno testirano je upoređeno sa drugim metodama za rešavanje velikih zagonetki. Uvođenjem vrlo jednostavnih pravila preprocesiranja brzina generisanja se dodatno poboljšava. Unapređen je postojeći algoritam za generisanje velikih sudoku zagonetki.

Izvorni kod implementacija svih sistema kao i ulazne datoteke i instance korišćene u eksperimentima (ali bez korišćenih rešavača drugih autora, zbog specifičnog licenciranja) su dostupni na veb-strani autora teze: `http://jason.matf.bg.ac.rs/~mirkos`.

1.3 Organizacija teze

U okviru glave 2 dat je pregled oblasti CP: prikazane su metode rešavanja problema CSP/COP, sistemi koji rešavaju te probleme kao i njihovi ulazni formati. Poseban akcenat stavljen je na deo koji se odnosi na rešavanje problema svođenjem na problem SAT. Predstavljena su postojeća i prikazana nova hibridna kodiranja za ovo svođenje i potvrđena je njihova efikasnost eksperimentalnim rezultatima. Dat je dokaz ispravnosti jednog od kodiranja koji nije postojao u literaturi. Prikazan je sistem *meSAT*, koji predstavlja prvi sistem koji omogućava odabir između većeg broja kodiranja pri svođenju na SAT.

U okviru glave 3, prvo je dat pregled aktuelnih tehnika mašinskog učenja kao i portfolija za rešavanje problema SAT i CSP. Predstavljen je portfolio ArgoCSP-kNN i pokazano je da je konkurentan najefikasnijim savremenim portfolijima. Demonstrirana je efikasnost portfolija kako u odabiru kodiranja tako i u odabiru rešavača. Na kraju ove glave predstavljen je portfolio pristup koji je zasnovan na kratkom treniranju.

U glavi 4 se bavimo primenama CSP rešavača. Jedna primena je njihovo korišćenje u rešavanju problema raspoređivanja kontrolora letetnja. Predstavljena su 3 modela ovog problema kao i tehnike korišćene u traženju optimalne vrednosti. Prikazani su i rezultati izvršenih eksperimenata. Druga primena se odnosi na generisanje i rešavanje sudoku zagonetki. Napravljen je pregled postojećih tehnika za rešavanje i generisanje velikih sudoku zagonetki pomoću računara. Izvršeni su eksperimenti sa ciljem da se uporedi rešavanje velikih zagonetki svođenjem na SAT sa drugim savremenim pristupima. Predstavljena je i testirana modifikacija postojećeg algoritma za generisanje velikih i teških zagonetki.

U glavi 5 je dat pregled rezultata ove teze i zaključci do kojih se došlo. Potom su predstavljene i neke ideje koje predstavljaju potencijalno dobre pravce za dalji naučni rad.

Glava 2

Rešavanje problema CSP svođenjem na problem SAT

2.1 Opis problema CSP

Prva faza u rešavanju nekog problema je *modeliranje* tog problema (u ovom tekstu se misli na predstavljanje u obliku problema CSP). U toj fazi je potrebno odrediti *model problema*, tj. odrediti koje promenljive se koriste, koji su domeni tih promenljivih i koja su ograničenja definisana nad tim promenljivama. Od više mogućih modela obično se bira jedan ili nekoliko njih za koje se smatra da su najpogodniji sa rešavanje.

U ovom poglavlju ćemo uvesti problem CSP i opisati različite vrste ograničenja koje omogućavaju da se problemi modeliraju na veliki broj različitih načina. Potom ćemo dati nekoliko primera problema CSP i prikazati podskup klase problema CSP koji je u fokusu ove teze. Zatim ćemo definisati optimizacionu verziju problema CSP. Na kraju ćemo predstaviti jezike za modeliranje problema CSP.

2.1.1 Definicija problema CSP

Definicija 1. Problem CSP *je trojka* (X, D, C)*, ako važe sledeći uslovi:*

1. *Skup* $X = \{x_1, \ldots, x_n\}$ *je konačan skup promenljivih.*

2. *Domen promenljive je skup vrednosti koje toj promenljivoj mogu da se dodele. Skup* $D = \{D_1, \ldots, D_n\}$ *je skup domena svih promenljivih, tj. važi da* $(x_1, \ldots, x_n) \in D_1 \times \ldots \times D_n$*.*

3. *Skup C je konačan skup ograničenja koji određuje koje vrednosti promenljive mogu da uzmu. Svako od ograničenja je definisano na nekom podskupu skupa svih promenljivih $\{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ $(i_1, \ldots, i_k \in \{1, \ldots, n\})$ i predstavlja podskup Dekartovog proizvoda $D_{i_1} \times \ldots \times D_{i_k}$. Ako je $k = 1$ kažemo da je ograničenje* unarno*, a ako je $k = 2$ kažemo da je ograničenje* binarno*.*

Kaže se da n-torka $A = (a_1, \ldots, a_n)$, $A \in D_1 \times \ldots \times D_n$, zadovoljava ograničenje *definisano na nekih k promenljivih $(1 \leq k \leq n)$, ako projekcija A na tih k promenljivih daje k-torku koja pripada tom ograničenju.* Rešenje *problema CSP je n-torka $A = (a_1, \ldots, a_n)$, $A \in D_1 \times \ldots \times D_n$, koja zadovoljava sva ograničenja tog problema. Dakle, potrebno je da vrednosti promenljivih pripadaju odgovarajućim domenima i da zadovoljavaju sva ograničenja problema. Pod* rešavanjem problema CSP *se podrazumeva postupak utvrđivanja da li problem ima rešenja i ako ima, pronalaženje jednog ili svih rešenja tog problema. Ako problem ima barem jedno rešenje kaže se da je* konzistentan *ili* zadovoljiv*, a ako nema rešenja kaže se da je* nekonzistentan *ili* nezadovoljiv*.*

Domeni promenljivih su obično skupovi brojeva, ali u opštem slučaju domen može biti i skup nekih simbola. Na primer, domen može biti i skup imena osoba, skup reči, skup geometrijskih objekata, itd.

Problem CSP je *normalizovan* ako za svaki podskup skupa svih promenljivih postoji najviše jedno ograničenje definisano na tom podskupu. U ovoj tezi ćemo probleme koje razmatramo smatrati normalizovanim – u slučaju da postoji više ograničenja definisanih na nekom podskupu, može se napraviti njihov presek i na taj način dobiti samo jedno ograničenje. Na primer, ako su na nekoj promenljivoj x_i definisana ograničenja $x_i \in [0, 3]$ i $x_i \in [2, 10]$, onda se ova dva ograničenja mogu zameniti ograničenjem $x_i \in [2, 3]$. Ovo nam omogućava da bez bojazni od višeznačnosti za unarno ograničenje definisano na promenljivoj x_i koristimo oznaku C_i, a za binarno ograničenje definisano na promenljivama x_i i x_j koristimo oznaku $C_{i,j}$ (ili $C_{j,i}$ jer je $C_{i,j} = C_{j,i}$).

Vrste ograničenja. Definicija 1 uvodi ograničenja koja se zovu *ekstenzionalna ograničenja* (eng. *extensional constraints*). Ova ograničenja se zadaju pomoću skupa kombinacija dozvoljenih/zabranjenih vrednosti promenljivih koje ograničavaju. Kod modeliranja problema kao problema CSP koriste se i *intenzionalna ograničenja* (eng. *intensional constraints*). Ova ograničenja se zadaju kao iskazne formule koje

se grade primenom iskaznih veznika ($\wedge$, $\vee$, $\oplus$, $\rightarrow$, $\leftrightarrow$) na atomičke formule koje se dobijaju primenom relacijskih simbola ($\leq$, $<$, $\geq$, $>$, $=$, $\neq$) na izraze (termove), koji su dobijeni primenom funkcijskih simbola (na primer, $+$, $-$, $*$, $/$, $\%$) na promenljive i konstante. Promenljive i konstante mogu biti različitih tipova (na primer, celobrojnog, realnog). Svako intenzionalno ograničenje se može transformisati u ekstenzionalno ograničenje, i obrnuto.

Primer 1. *Problem CSP sadrži celobrojne promenljive x_1 i x_2 sa domenima $D_1 = D_2 = \{1, 2, 3\}$ i potrebno je zadovoljiti dva ograničenja. Prvo je ekstenzionalno ograničenje $\{(1,1), (1,2), (1,3), (2,2), (2,3), (3,3)\}$ (može se zadati i u obliku intenzionalnog ograničenja $x_1 \leq x_2$). Drugo je intenzionalno ograničenje $(x_1 > x_2) \vee (x_1 + x_2 = 6)$ (može zadati i u kao ekstenzionalno ograničenje $\{(2,1), (3,1), (3,2), (3,3)\}$). Primetimo da ovaj problem nije normalizovan, jer su nad parom promenljivih (x_1, x_2) definisana dva ograničenja. Jedino rešenje ovog problema je $x_1 = 3$, $x_2 = 3$.*

Treća vrsta ograničenja su *globalna ograničenja*. Njima je posvećena posebna pažnja u ovom radu pa su detaljnije opisana u poglavlju koje sledi.

2.1.2 Globalna ograničenja

Globalno ograničenje (eng. *global constraint*)[1] je ograničenje koje definiše odnos između nefiksiranog broja promenljivih. Postoje dve glavne koristi u odnosu na intenzionalna i ekstenzionalna ograničenja. Prva je da se korišćenjem globalnih, umesto druge dve vrste ograničenja (globalna ograničenja se uvek mogu zameniti ekstenzionalnim ograničenjima), pri modeliranju problema obično dobija jednostavniji i razumljiviji model. Druga korist je da se mogu razviti posebni algoritmi za globalna ograničenja čime se često značajno poboljšava efikasnost u rešavanju problema ([106]). Navešćemo primere nekoliko globalnih ograničenja koja se često koriste. Ova ograničenja mogu biti zadata nad proizvoljnim izrazima (koji su često konstante, tj. promenljive).

Ograničenje *all-different*. Ovo je najpoznatije globalno ograničenje. Potrebno je da svi argumenti (izrazi) nad kojima je zadato ovo ograničenje uzmu različite vrednosti, tj., *all-different* $(e_1, \ldots, e_n)$ postavlja uslov da je $e_i \neq e_j$ za bilo koje

[1] Katalog globalnih ograničenja [12] je dostupan na: `http://www.emn.fr/z-info/sdemasse/gccat`

$i \neq j$, $i, j \in \{1, \ldots, n\}$. Broj n u slučaju ovog kao i ostalih globalnih ograničenja nije fiksiran, tj. n može biti bilo koji prirodan broj.

Ograničenje *nvalue*. Ograničenje $nvalue(\{e_1, \ldots, e_n\}, e)$ je zadovoljeno ako izrazi $e_1, \ldots, e_n$ uzimaju broj različitih vrednosti jednak vrednosti izraza e. Na primer, ograničenje $nvalue(\{x_1, x_2, x_3\}, 2)$ (gde je $e_i = x_i$ i $e = 2$) je zadovoljeno ako promenljive x_1, x_2 i x_3 uzimaju ukupno dve različite vrednosti. Specijalan slučaj kada je $e = n$ predstavlja *all-different* ograničenje (svi izrazi moraju uzeti međusobno različite vrednosti).

Ograničenje *count*. Ograničenje $count(\{e_1, \ldots, e_n\}, e)\ R\ e'$ predstavlja uslov da broj pojavljivanja vrednosti nekog izraza e u skupu vrednosti izraza $e_1, \ldots, e_n$ bude u odgovarajućoj aritmetičkoj relaciji R ($=$, $\neq$, $\leq$, $<$, $\geq$, $>$) sa nekim izrazom e'. Na primer, ograničenje $count(\{x_1, x_2, x_3, x_4\},\ 5) > 3$ (gde je $e = 5$, $e_i = x_i$, $n = 4$, $e' = 3$, a relacija je $>$) je zadovoljeno ako se vrednost 5 pojavljuje više od 3 puta (tj. bar 4 puta) u skupu promenljivih $\{x_1, x_2, x_3, x_4\}$. Ovo znači da sve promenljive x_1, x_2, x_3 i x_4 moraju da uzmu vrednost 5.

2.1.3 Primeri problema CSP

Primeri koji slede predstavljaju modifikacije primera iz knjige "Principles of Constraint Programming" autora Kžištofa Apta [4].

Primer 2 (*Problem n kraljica.*)**.** *Potrebno je rešiti problem postavljanja n kraljica na šahovsku tablu dimenzija $n \times n$, tako da među njima ne postoje dve kraljice koje se međusobno napadaju. Postoji više načina modeliranja ovog problema, a mi ovde prikazujemo tri.*

Prvi model. *U prvom modelu, i-toj kraljici se pridružuje i-ta kolona i promenljiva x_i sa domenom $\{1, \ldots, n\}$ koja označava u kom redu se nalazi i-ta kraljica. Dakle, skup promenljivih je $X = \{x_1, \ldots, x_n\}$, a odgovarajući domeni su $D_1 = \ldots = D_n = \{1, \ldots, n\}$. Za svake dve različite kolone i i j ($1 \leq i < j \leq n$) moraju biti zadovoljena ograničenja $x_i \neq x_j$ (nisu u istom redu), $x_i + i \neq x_j + j$ i $x_i - i \neq x_j - j$ (ne napadaju se ni po jednoj od dijagonala).*

Drugi model. *Ovaj model je isti kao prvi model, ali se umesto uvođenja ograničenja da se dve po dve vrednosti razlikuju, uvode globalna ograničenja all-different $(x_1, \ldots, x_n)$, all-different $(x_1 + 1, \ldots, x_n + n)$ i all-different $(x_1 - 1, \ldots, x_n - n)$.*

Treći model. *U trećem modelu, svakom polju (i, j) table se pridružuje promenljiva $y_{i,j}$ sa domenom $\{0, 1\}$ koja uzima vrednost 1 akko se na polju (i, j) nalazi neka kraljica. Za svaku vrstu $i \in \{1, \ldots, n\}$ (kolonu $j \in \{1, \ldots, n\}$) mora biti zadovoljeno ograničenje $y_{i,1} + \ldots + y_{i,n} = 1$ ($y_{1,j} + \ldots + y_{n,j} = 1$) – u svakoj vrsti (koloni) nalazi se tačno jedna kraljica. Takođe, uvode se i ograničenja koja obezbeđuju da se na svakoj (bilo manjoj ili većoj) dijagonali nalazi najviše jedna kraljica (na primer, $y_{1,1} + \ldots + y_{n,n} \leq 1$ za glavnu dijagonalu).*

U navedenim modelima su korišćena intenzionalna i globalna ograničenja, što je u ovom slučaju mnogo kompaktniji i jasniji način navođenja ograničenja u odnosu na korišćenje ekstenzionalnih ograničenja.

Interesantno je da postoji način da se reši ovaj problem popunjavanjem tačno određenih polja za bilo koje n [15]. Uprkos tome, ovaj problem ostaje jedan od centralnih u oblasti programiranja ograničenja jer se lako kreiraju različiti modeli, pogodan je za prikaz i razumevanja razvijenih algoritama, ali i za poređenje različitih sistema i metoda rešavanja [69].

Primer 3 (*SEND + MORE = MONEY.*)**.** *Potrebno je slova zameniti međusobno različitim ciframa tako da bude ispunjeno:*

$$\begin{array}{r} SEND \\ +\ MORE \\ \hline = MONEY \end{array}$$

Promenljive S, E, N, D, M, O, R, Y imaju domene $\{0, \ldots, 9\}$. Jedan način da se uvedu ograničenja ovog problema je:

$$\begin{gathered} S \neq 0, \\ M \neq 0, \\ \textit{all-different}\ (S, E, N, D, M, O, R, Y), \\ 1000 \cdot S + 100 \cdot E + 10 \cdot N + D \\ +1000 \cdot M + 100 \cdot O + 10 \cdot R + E \\ = 10000 \cdot M + 1000 \cdot O + 100 \cdot N + 10 \cdot E + Y. \end{gathered}$$

2.1.4 Konačni linearni CSP

Iako je data definicija 1 problema CSP veoma opšta i pokriva veliki broj vrlo raznolikih problema (na primer, rešavanje sistema jednačina, nalaženje nula polinoma, itd.), metode za rešavanje problema CSP su obično efikasne samo za podskup problema CSP: promenljive mogu biti samo bulovske i celobrojne, celobrojne promenljive su konačnog domena i dozvoljeni su samo linearni izrazi nad celobrojnim promenljivama. U ovoj tezi fokus je upravo na tom specijalnom obliku problema CSP koji se u praksi najčešće rešava.

Definicija 2. Linearni izrazi *nad konačnim skupom celobrojnih promenljivih (konačnih domena) X su algebarski izrazi oblika* $\sum_{i=0}^{m-1} a_i x_i$, *gde* $\forall i \in \{0, \ldots, m-1\}$ $x_i \in X$, $a_i \in \mathbb{Z}$.

Konačni linearni CSP u KNF-u *je petorka* (X, L, U, B, C), *gde važi:*

1. *X je konačan skup celobrojnih promenljivih.*
2. *$L : X \mapsto \mathbb{Z}$ i $U : X \mapsto \mathbb{Z}$ su donja i gornja granica celobrojnih promenljivih i ove granice određuju domen $D(x)$ svake promenljive x.*
3. *B je skup iskaznih promenljivih.*
4. *C je konačan skup klauza (nad X i B). Klauze su disjunkcije literala, pri čemu su literali elementi unije skupova B, $\{\neg p \mid p \in B\}$ i $\{e \leq c \mid e$ je linearni izraz nad X, $c \in \mathbb{Z}\}$.*

KNF je skraćenica za konjunktivnu normalnu formu *(eng.* conjunctive normal form*) i označava da je skup C zadat kao konačan skup klauza.*

Rešenje *konačnog linearnog problema CSP u KNF-u je dodela iskaznih vrednosti iskaznim promenljivama i celobrojnih vrednosti celobrojnim promenljivama tako da pripadaju odgovarajućim domenima promenljivih i da su posle zamene promenljivih dodeljenim vrednostima sve klauze iz C zadovoljene.*

Primer 4. *Rešenje konačnog linearnog problema CSP (X, L, U, B, C), gde je $X = \{x_1, x_2, x_3\}$, $L = \{x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 2\}$, $U = \{x_1 \mapsto 2, x_2 \mapsto 4, x_3 \mapsto 3\}$, $B = \{p\}$, $C = \{p \vee x_1 + x_3 \leq 4, \neg p \vee x_3 + (-1) \cdot x_1 \leq 0, x_1 \leq 1 \vee 2 \cdot x_2 \leq 4\}$, je dodela $\{p \mapsto \bot, x_1 \mapsto 1, x_2 \mapsto 3, x_3 \mapsto 2\}$.*

U primenama se obično ulazna sintaksa proširuje tako da su omogućeni nekontinualni domeni, formule sa proizvoljnom bulovskom strukturom (ne samo KNF), a

literali se formiraju primenom i drugih aritmetičkih relacija (na primer, $<$, $\leq$, $\geq$, $>$, $=$) i drugih aritmetičkih operacija (na primer, celobrojno deljenje, ostatak pri deljenju). Kao što je već rečeno u poglavlju 2.1.1, ova ograničenja se zovu intenzionalna ograničenja. Na sličan način kao u tom poglavlju se uvode i ekstenzionalna i globalna ograničenja. Sve navedene vrste ograničenja se mogu prevesti u konačni linearni problem CSP u KNF-u, ali se obično postiže veća efikasnost ako se u okviru rešavača problema CSP implementiraju posebno razvijeni algoritmi koji direktno obrađuju svako od ograničenja.

2.1.5 Definicija problema COP

Definicija 3. Problem optimizacije uz ograničenja *ili* problem uslovne optimizacije *(eng.* Constraint Optimization Problem*), skraćeno COP, je problem nalaženja* n*-torke* $(x_1, \ldots, x_n) \in D_1 \times \ldots \times D_n$ *koja zadovoljava skup ograničenja* C *i daje maksimalnu vrednost* funkcije cilja *ili* kriterijumske funkcije *(eng.* objective function*)* $f : D_1 \times \ldots \times D_n \to \mathbb{R}$*. Svaka takva* n*-torka se zove* optimalno rešenje *ili* optimum *maksimizacionog/minimizacionog problema COP.*

Problem COP je po svojoj definiciji vrlo sličan problemu CSP: razlika je jedino u postojanju funkcije cilja. Ipak, metode za rešavanje ove dve vrste problema se u praksi značajno razlikuju. Sam problem COP moze se definisati kao problem CSP, ako se maksimizacija/minimizacija funkcije cilja smatra jednim od ograničenja problema CSP. Međutim, neuporedivo efikasnije metode za rešavanje problema COP se konstruišu ako se funkcija cilja razmatra odvojeno od ostalih ograničenja.

Bez gubitka opštosti može se pretpostaviti da je uvek potrebno naći maksimalnu vrednost, jer se u slučaju traženja minimalne vrednosti može formirati funkcija cilja koja bilo kojoj kombinaciji vrednosti promenljivih $(b_1, \ldots, b_n)$ umesto vrednosti $f(b_1, \ldots, b_n)$ pridružuje njoj suprotnu vrednost $-f(b_1, \ldots, b_n)$.

Primer 5 (*Problem ranca [135, 39].*)**.** *Dato je* n *predmeta kojima su pridružene zapremine* $z_1, \ldots, z_n$ *i vrednosti* $v_1, \ldots, v_n$*. Potrebno je pronaći podskup predmeta koji u zbiru imaju maksimalnu vrednost i koji staju u ranac zapremine* Z*. Problem se može modelirati tako što se uvedu promenljive* $x_1, \ldots, x_n$ *sa domenom* $\{0, 1\}$ *i ograničenje* $\sum_{i=1}^{n} z_i x_i \leq Z$*. Potrebno je naći rešenje tako da vrednost* $\sum_{i=1}^{n} v_i x_i$ *bude maksimalna, tj.* $\sum_{i=1}^{n} v_i x_i$ *je funkcija cilja ovog problema i treba je maksimizovati.*

U fokusu ove teze su problemi CSP. U delovima teksta koji se odnose na optimizacionu verziju ovog problema, to će biti posebno naglašeno.

2.1.6 Jezici za modeliranje problema CSP

Postoji veliki broj različitih jezika u kojima se mogu formulisati problemi CSP. Pri razmatranju jezika govorićemo o jezicima različitih *nivoa*, odnosno o jezicima *niskog nivoa* i *visokog nivoa*. Kod jezika niskog nivoa problemi se obično predstavljaju navođenjem svih promenljivih, njihovih domena i ograničenja. Ovime se uglavnom dobijaju velike datoteke koje nisu čitljive i kod kojih model nije odvojen od konkretnih ulaznih podataka. Obično osoba koja modelira problem u jeziku niskog nivoa mora imati dodatno znanje o tome kako modelirati problem da bi on bio pogodan za rešavanje. Prednost jezika niskog nivoa je upravo u tome što se njihovim korišćenjem, uz dodatni uloženi napor, može postići velika efikasnost u rešavanju problema. Kod jezika visokog nivoa omogućen je kompaktniji i obično čitljiviji zapis (na primer, korišćenje `for` petlji, korišćenje nizova, skupova i skupovnih operacija), moguće je odvojiti model od podataka, itd. Prednost jezika visokog nivoa je i da se u njima obično lako modeliraju problemi. Za pojedine jezike se po nekim njihovim osobinama može reći da su visokog nivoa, a po nekim drugim osobinama da su niskog nivoa, tj. ne može se za svaki jezik utvrditi da isključivo pripada jednoj ili drugoj grupaciji jezika. Naveśćemo nekoliko najpoznatijih jezika u kojima se mogu predstaviti problemi CSP. Svi primeri koji slede odgovaraju drugom modelu iz primera 2 (n kraljica).

2.1.6.1 MiniZinc, FlatZinc, XCSP i Sugar jezik

MiniZinc[2] [98] je u trenutku pisanja ovog teksta najrasprostranjeniji jezik u CP zajednici. Ovaj jezik visokog nivoa predstavlja podskup jezika visokog nivoa Zinc. Podržana su sva ograničenja koja su opisana u poglavlju 2.1.4 kao i veliki broj globalnih ograničenja. Dodatno, ovaj jezik omogućava korišćenje nizova, skupova, realnih brojeva kao i veliki broj ugrađenih predikata. Obično se MiniZinc model (datoteka sa ekstenzijom .mzn) i podaci (datoteka sa ekstenzijom .dzn)[3] prevode u model na jeziku FlatZinc, koji je niskog nivoa. U tom jeziku se navode promenljive i ograničenja, kao i funkcija cilja u slučaju optimizacionog problema. Globalna

[2] `http://www.minizinc.org`

[3] Podaci se mogu navesti i u okviru .mzn datoteke ali je u praksi preporučljivo odvojiti model od podataka.

```
int: n;
array [1..n] of var 1..n: x;

include „alldifferent.mzn";

constraint alldifferent(x);
constraint alldifferent([ x[i] + i | i in 1..n]);
constraint alldifferent([ x[i] - i | i in 1..n]);

solve :: int_search(x, first_fail, indomain_min, complete) satisfy;
```

Slika 2.1: Problem n kraljica u MiniZinc jeziku. Ograničenje *all-different* je jedno od mnogobrojnih globalnih ograničenja koje je podržano ovim jezikom. Poslednji red sadrži anotacije koje predstavljaju preporuku za metodu traženja rešenja.

ograničenja se pri prevođenju u FlatZinc transformišu u druge dve vrste ograničenja. Rešavači obično prihvataju datoteku u jeziku FlatZinc ali mogu da naznače način za prevođenje ograničenja modela iz MiniZinc-a u FlatZinc, da bi dobijena FlatZinc datoteka bila pogodnija za njihov pristup rešavanju problema. Model može ali ne mora da sadrži anotacije koje rešavaču sugerišu koji način rešavanja da primeni. Postoji i takmičenje MiniZinc Challenge[4] u kome se koriste instance u MiniZinc formatu. Na slici 2.1 je predstavljen model problema n kraljica u MiniZinc jeziku. Na osnovu modela i vrednosti n koja se nalazi u odvojenoj datoteci, moguće je pomoću dostupne alatke u MiniZinc distribuciji automatski generisati datoteku u FlatZinc formatu.

Jezik XCSP [109] je jezik niskog nivoa, poput jezika FlatZinc. Podržana su sva ograničenja koja su opisana u poglavlju 2.1.4 (za razliku od jezika FlatZinc podržana su i globalna ograničenja). Ovaj jezik je više puta korišćen kao ulazni jezik rešavača koji su učestvovali na internacionalnim takmičenjima CSP rešavača[5]. Verzija jezika koja je u najširoj upotrebi je 2.1. Razvijena je i verzija 3.0 ovog jezika (predstavljena na konferenciji CP 2015) ali u trenutku pisanja teksta ona još nije bila u široj upotrebi. Jezik se zasniva na XML reprezentaciji problema CSP. Na slici 2.2 je predstavljena specifikacija problema 4 kraljice. Informacija o broju kraljica nalazi se u samoj specifikaciji, pa je za drugačiji broj kraljica potrebno generisati drugačiju datoteku (ovo se može automatizovati pisanjem programa-generatora koji će kao argument primiti broj kraljica). Neophodno je definisati i koristiti predikate

[4] `http://www.minizinc.org/challenge.html`
[5] `http://www.cril.univ-artois.fr/CPAI09`

```
<domains nbDomains=„1”>
<domain name=„D1” nbValues=„4”>1..4</domain>
</domains>

<variables nbVariables=„4”>
<variable name=„x1” domain=„D1”/>
<variable name=„x2” domain=„D1”/>
<variable name=„x3” domain=„D1”/>
<variable name=„x4” domain=„D1”/>
</variables>

<predicates nbPredicates=„1”>
<predicate name=„P1”><parameters>int X1 int X2 int X3 int X4</parameters>
<expression><functional>ne(add(X1,X2),add(X3,X4))
</functional></expression></predicate>
</predicates>

<constraints nbConstraints=„84”>
<constraint name=„C1” arity=„2” scope=„x1 x2” reference=„P1”>
<parameters>x1 1 x2 2</parameters></constraint>
<constraint name=„C2” arity=„2” scope=„x1 x2” reference=„P1”>
<parameters>x1 -1 x2 -2</parameters></constraint>
<constraint name=„C3” arity=„2” scope=„x1 x2” reference=„P1”>
<parameters>x1 0 x2 0</parameters></constraint>

... (ista tri ograničenja za svaka dva para promenljivih)
</constraints>
```

Slika 2.2: Problem 4 kraljice u XCSP jeziku (izbačeno je nekoliko redova zaglavlja sa početka). Moguće je koristiti ograničenje *all-different*, ali ono mora biti definisano nad promenljivama, pa su ovde jednostavnosti radi korišćene nejednakosti.

(ograničenja koja korisnik definiše nad već postojećim ograničenjima), a predikati koji se odnose na globalna ograničenja su dostupni bez navođenja definicije.

Sistem Sugar [124], koji rešava problem CSP svođenjem na SAT, ima svoj jezik (nadalje ćemo ovaj jezik zvati *Sugar jezik*). Skup ograničenja koja su podržana ovim jezikom je isti kao i kod XCSP jezika, ali je Sugar jezik znatno čitljiviji. Na slici 2.3 je predstavljena specifikacija problema 4 kraljice. Nije neophodno koristiti predikate kao u slučaja jezika XCSP, a ne postoje ni etikete.

Slike 2.1, 2.2 i 2.3 pokazuju da je MiniZinc jezik višeg nivoa od druga dva jezika – problemi se zadaju na kompaktniji način i specifikacije su (obično) čitljivije.

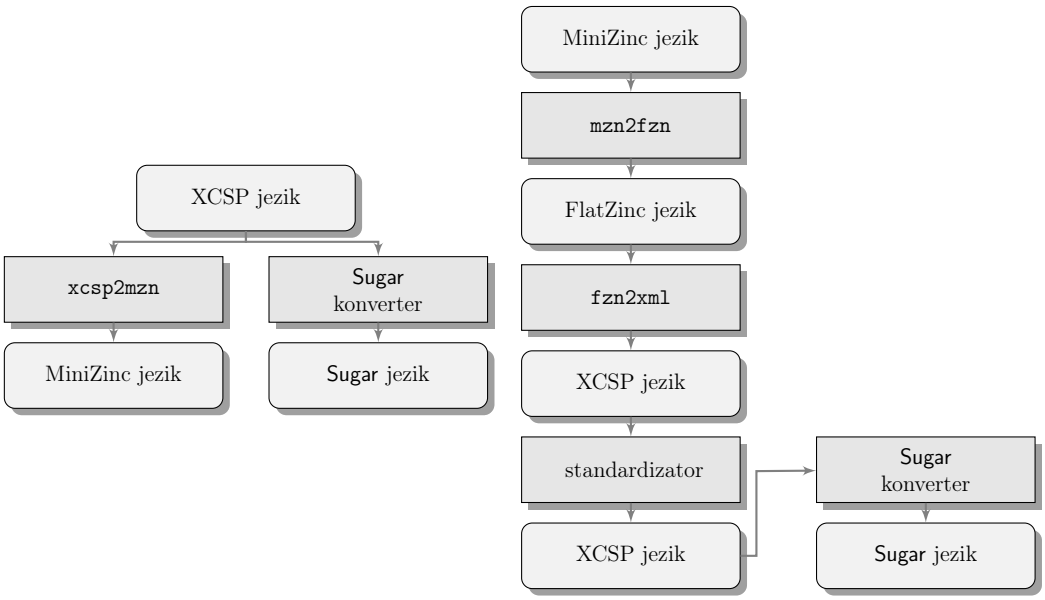
```
(int x1 1 4) (int x2 1 4) (int x3 1 4) (int x4 1 4)
(alldifferent x1 x2 x3 x4)
(alldifferent (+ x1 1) (+ x2 2) (+ x3 3) (+ x4 4))
(alldifferent (- x1 1) (- x2 2) (- x3 3) (- x4 4))
```

Slika 2.3: Problem 4 kraljice u Sugar jeziku. Ograničenje *all-different* je jedno od globalnih ograničenja koje je podržano ovim jezikom.

Međutim, kada se specifikacija u jeziku MiniZinc prebaci u jezik FlatZinc, dobija se definicija problema koja je sličnog nivoa kao i XCSP i Sugar jezik. Ovakvu definiciju problema primaju rešavači. Kod jezika XCSP i Sugar jezika se kompaktnost i čitljivost postižu pomoću posebno razvijenih programa-generatora napisanih u proizvoljnom programskom jeziku (na primer, C, C++, Java). Bitna prednost jezika MiniZinc je da on podržava opštiji oblik problema CSP od druga dva jezika – dozvoljeno je korišćenje realnih brojeva, nizova, skupova i šireg skupa globalnih ograničenja.

U trenutku pisanja teze najveći broj sistema kao ulaz koristi jezik FlatZinc uz navođenje pravila za prevod ograničenja iz jezika MiniZinc u ovaj jezik – ova pravila omogućavaju da se prevođenje izvrši tako da dobijena FlatZinc instanca bude pogodna za odgovarajući sistem. XCSP je takođe široko rasprostranjen ali se takmičenja rešavača koji koriste ovaj jezik ne održavaju već nekoliko godina, pa se u nekim slučajevima ne razvijaju ni nove verzije rešavača koji koriste ovaj jezik. Ipak, zbog velikog broja XCSP instanci korišćenih na ranijim takmičenjima, ovaj jezik se i dalje dosta koristi. Sugar jezik koristi nekoliko sistema i njihove verzije se uglavnom redovno ažuriraju.

Postoje alatke koje prevode specifikacije iz jednog od navedenih jezika (MiniZinc, XCSP, Sugar) u bilo koji od preostala dva. Postupak prevođenja iz XCSP jezika u preostala dva jezika i iz MiniZinc jezika u preostala dva jezika je prikazan na slici 2.4. Program `xcsp2mzn`, koji je dostupan na veb-strani sistema MiniZincprevodi specifikaciju iz XCSP jezika u MiniZinc. Sistem Sugar u sebi sadrži prevodilac iz XCSP-a u Sugar jezik. MiniZinc specifikacija se prevodi u FlatZinc pomoću alatke `mzn2fzn`, a iz ovog jezika u XCSP pomoću alatke `fzn2xml` (obe alatke su uključena u MiniZinc distribuciju). XCSP instance se pomoću alatke koji su razvili Amadini sa koautorima [2] mogu standardizovati, kako bi zadovoljavale uslove koji su zahtevani na prethodno održanim takmičenjima CSP rešavača. Koliko nam je poznato, ne postoji alatka za prevođenje iz Sugar jezika u ostala dva jezika. Ovo nije bitno



Slika 2.4: Konverzije između različitih ulaznih jezika.

ograničenje pošto je najveći broj javno dostupnih CSP instanci u jezicima MiniZinc i XCSP. Bitno je naglasiti da se sve Sugar i XCSP instance mogu prevesti u MiniZinc ali obrnuto ne važi, jer MiniZinc ima veću izražajnost (na primer, podržava realne brojeve, a oni nisu podržani jezikom XCSP).

2.1.6.2 Ostali jezici

Još jedan način za predstavljanje problema CSP je pomoću proširenja sintakse logičkih programskih jezika. Sintaksa ovih jezika kod kojih specifikacija problema sadrži bazu činjenica se u tom slučaju proširuje da bi omogućila specfikaciju ograničenja (npr. globalnih ograničenja). Najčešće se vrši proširenje PROLOG-a, a najpoznatiji jezici ovog tipa su NPSpec [28], `SWI-Prolog` [133], `CHIP` [13], kao i jezici koje koriste sistemi `BProlog` [136] i `ECLiPSe` [5].

Postoje i jezici visokog nivoa koji su pre svega razvijeni za rešavanje problema matematičkog programiranja ali pružaju podršku za korišćenje CP tehnika. Najpoznatiji su OPL [61] i AMPL [52]. OPL je prvi jezik koji se našao u široj upotrebi i koji je omogućio rešavanje problema matematičkog programiranja i programiranja ograničenja. On omogućava da se zadaju i detalji u vezi sa načinom nalaženja rešenja.

Još neki primeri CP jezika su `OZ` [117] (kombinuje koncepte različitih programskih

paradigmi), jezik URSA [69] koji predstavlja kombinaciju imperativnog i deklarativnog jezika, itd.

2.2 Pregled metoda rešavanja problema CSP

Pri rešavanju problema CSP, prvo se razmatra koja metoda će se upotrebiti u rešavanju i odlučuje da li će se primeniti neka metoda koja je posebno razvijena za tu vrstu problema ili neka opšta metoda. Skoro po pravilu se bolji rezultati dobijaju kada se primene metode posebno razvijene za odgovarajuću vrstu problema. Kod velikog broja problema se javljaju ograničenja koja imaju različitu formu i odgovaraju raznim oblastima, pa se u tom slučaju moraju koristiti metode opštijeg tipa za rešavanje problema CSP. U ovom delu ćemo se fokusirati na ove opšte metode.

Pretraga rešenja problema CSP može se obaviti na dva načina [111].

Potpuno nabrajanje ili *potpuno pretraživanje* (eng. *enumeration*) je najjednostavnija metoda za traženje rešenja problema CSP. Obuhvata prolazak kroz sve n-torke Dekartovog proizvoda domena promenljivih i proveru za svaku od tih n-torki da li jeste rešenje problema. Ova metoda je u praksi vrlo neefikasna i zato se za rešavanje problema CSP obično koriste efikasnije metode.

Posredno nabrajanje ili *posredno pretraživanje* je grupa metoda kojom se ispituje samo relativno mali broj n-torki Dekartovog proizvoda domena promenljivih, a ne sve n-torke. Koriste se različite tehnike kako bi se smanjio broj rešenja koja se istpituju, ali se među tim rešenjima uvek nalazi tačno rešenje, ako postoji. U ovoj tezi razmatramo tri vrste posrednog nabrajanja: *pretraga sa povratkom* (eng. *backtracking*), *propagiranje ograničenja* (eng. *constraint propagation*) i *lokalna pretraga* (eng. *local search*). Ove tehnike se obično kombinuju, kako bi se problemi efikasnije rešili korišćenjem dobrih strana svake od tehnika.

Algoritmi za rešavanje problema CSP mogu biti *potpuni* ili *nepotpuni*. Potpuni (zovu se i *egzaktni* ili *sistematični*) algoritmi garantuju da će rešenje biti pronađeno, ako postoji, a da će u suprotnom biti pokazano da rešenje ne postoji. Ovoj grupi pripada pretraga sa povratkom. Nepotpuni (zovu se i *heuristički* ili *nesistematični*) algoritmi ne mogu da pokažu da problem CSP nema rešenja. Sa druge strane, ovi algoritmi su često vrlo efikasni u nalaženju rešenja kada ono postoji. Lokalna pretraga i propagiranje ograničenja su primeri algoritama ovog tipa.

Iako je kombinacija pretrage i propagiranja najčešći pristup rešavanju problema CSP, postoje i drugi popularni pristupi. Programiranje ograničenja se u početku

uglavnom razvijalo kao *logičko programiranje uz ograničenja* (eng. *constraint logic programming*) [5], a ovakav pristup rešavanju problema CSP je i dalje veoma popularan. Kao što je već spomenuto u poglavlju 2.1.6, u ovom pristupu se sintaksa logičkih programskih jezika proširuje kako bi omogućila i specifikaciju nekih ograničenja koje nije moguće zadati u originalnim jezicima. Ipak, značajnije od proširenja sintakse jeste proširenje u vidu novih metoda rešavanja. Kombinuju se algoritmi logičkog programiranja (već postoji implementirana pretraga sa povratkom) sa efikasnim algoritmima za propagiranje ograničenja. Korisnik može da naznači vrstu pretrage koja se koristi u rešavanju, da naznači na koji način će biti obrađena ograničenja i da zada još neke parametre koji određuju tok rešavanja.

Pristup rešavanju koji je postao popularan u poslednjih par decenija je rešavanje problema CSP *svođenjem na druge probleme*, na primer, SAT [53], SMT [24]. Problem na koji se svodi problem CSP ćemo zvati *svedeni problem*. U ovom pristupu se obično najviše truda ulaže u postupak svođenja problema CSP u oblik pogodan za rešavače svedenih problema, a onda se na svedenim problemima koriste već gotovi rešavači tih problema. *Lenjo generisanje klauza* [103] kombinuje propagiranje ograničenja sa SAT rešavanjem i ovaj pristup se u nekim slučajevima pokazuje boljim u odnosu na klasično svođenje na SAT.

U naredna tri poglavlja ćemo opisati pretragu sa povratkom zajedno sa njenim modifikacijama, propagiranje ograničenja i lokalnu pretragu. Zatim će biti kratko opisana i jedna tehnika rešavanja problema COP. Potom će biti kratko opisani i postojeći softverski sistemi za rešavanje problema CSP. Što se tiče ostalih pristupa za rešavanje problema CSP, logičko programiranje uz ograničenja je kratko opisano u prethodnom pasusu i neće biti detaljno opisivano. Svođenje problema CSP na druge probleme i softverski sistemi koji koriste neko od svođenja su detaljno opisani u poglavlju 2.3.

2.2.1 Pretraga sa povratkom i njene modifikacije

Ovo je najčešće korišćena tehnika pretrage. Ova tehnika je potpuna, tj. postoji garancija da će se ona završiti sa odgovorom na pitanje da li je problem zadovoljiv ili ne; u slučaju zadovoljivosti ova tehnika će vratiti rešenje/skup rešenja problema CSP. Pseudokodovi algoritama u ovom poglavlju predstavljaju modifkacije algoritama koje je izložio Roman Bartak [11].

2.2.1.1 Naivna pretraga sa povratkom

Naivna pretraga sa povratkom (eng. *naive backtracking*) se naziva i *hronološka pretraga sa povratkom* (eng. *chronological backtracking*). Opisaćemo jednu jednostavnu varijaciju ovog algoritma za rešavanje problema CSP, a kasnije će biti razmatrane i razne njene modifikacije. Sam algoritam se može posmatrati kao obilazak u dubinu *stabla pretrage* (LKD obilazak, odnosno obilazak redosledom levo-koren-desno). Stablo pretrage se generiše kako pretraga napreduje i predstavlja izbore koji se ispituju u potrazi za rešenjem. Svaki čvor stabla je okarakterisan trenutnim domenima, ograničenjima i skupom dodeljenih vrednosti promenljivama, dok grane stabla odgovaraju dodelama vrednosti pojedinačnim promenljivama. Stablo se ne gradi eksplicitno, već implicitno – razmatranje stabla omogućava bolje razumevanje načina funcionisanja pretrage.

Svakom čvoru u stablu pretrage se pridružuje neka promenljiva koja se u trenutku posete čvora zove *tekuća promenljiva*. Obilaskom čvora se pokušava sa dodelom vrednosti tekućoj promenljivoj – svakoj dodeljenoj vrednosti odgovara grana ka novom čvoru koji se posećuje i u kome se pokušava sa dodelom vrednosti nekoj drugoj promenljivoj. Pošto je u pitanju obilazak u dubinu, prvo se formiraju grana i novi čvor i odmah se posećuje taj čvor, a tek kasnije se formira naredna grana i odgovarajući čvor, itd. Sve promenljive koje su pridružene čvorovima prethodnicima (direktnim precima) tekućeg čvora već imaju dodeljene vrednosti i njih zovemo *prošle promenljive*, a skup tih promenljivih zajedno sa dodeljenim vrednostima ćemo zvati *parcijalno rešenje*. Promenljive kojima još nisu dodeljene vrednosti zovemo *buduće promenljive*.

Ograničenja se koriste da se proveri da li čvor može potencijalno da vodi do rešenja problema CSP i da se izbegne formiranje (tj. izvrši odsecanje) podstabala koja ne sadrže rešenja. Ako u nekom čvoru nisu sve promenljive dobile vrednost ali se tekućoj promenljivoj ni na koji način ne može dodeliti vrednost tako da ograničenja ostanu zadovoljena (odgovarajući čvor nema ni jedno dete), onda tu situaciju zovemo *konflikt*, promenljivu kojoj nije moguće dodeliti vrednost *konfliktna promenljiva*, a sam čvor *konfliktni čvor*. U slučaju nailaska na konfliktni čvor, potrebno je izvršiti povratak u njegovog najbližeg prethodnika u kome nisu isprobane sve vrednosti tekuće promenljive i pokušati formiranje drugih grana iz tog prethodnika. Ako se u nekom trenutku pretragom dođe do čvora u kome su sve promenljive dobile vrednost (list stabla), onda je nađeno rešenje problema CSP. Ako je cilj pronaći samo jedno rešenje problema, onda u tom trenutku treba prekinuti algoritam, bez

konstrukcije ostatka stabla pretrage. Ako je cilj naći sva rešenja, onda se nastavlja sa konstrukcijom i obilaskom stabla.

Pseudokod algoritma naivne pretrage sa povratkom za rešavanje problema CSP dat je na slici 2.5. Predstavljena verzija algoritma traži samo jedno rešenje problema, a tekuća promenljiva označena je sa x. Pri početnom pozivu su sve promenljive u skupu `Buduće`, tj. nijedna promenljiva nema dodeljenu vrednost (ovo odgovara korenu stabla). U funkciji `BT` bira se promenljiva sa najmanjim indeksom koja nema dodeljenu vrednost i redom se pokušava sa dodeljivanjem različitih vrednosti iz domena te promenljive. Ako sva ograničenja koja se odnose samo na promenljive kojima su do tog trenutka dodeljene vrednosti i posle dodele ostaju zadovoljena (utvrđuje se pozivom funkcije `Proveri_konzistenciju`), generiše se novi rekurzivni poziv (pravi se grana stabla) u kome je izabrana promenljiva zajedno sa dodeljenom vrednošću prebačena u skup `Prošle`. Ako neko ograničenje posle dodele nije zadovoljeno, onda se parcijalno rešenje sigurno ne može produžiti do rešenja, pa se bez rekurzivnog poziva (bez formiranja grane stabla) pokušava sa narednom vrednošću promenljive. Ako se u pretrazi dođe do situacije kada je svaka promenljiva dobila vrednost (`Buduće` $= \emptyset$), pronađeno je rešenje problema (sva ograničenja su zadovoljena) – u ovom trenutku se redom završavaju rekurzivni pozivi i vraća se rešenje. Ako su u nekom čvoru sve vrednosti promenljive isprobane i ni za jednu se nije došlo do rešenja problema (ovu situaciju zovemo *neuspeh*), onda je potrebno vratiti se na poslednju promenljivu za koju nisu isprobane sve vrednosti – završetkom jednog ili više rekurzivnih poziva (penjanje uz stablo) – i probati sa dodelom nove vrednosti toj promenljivoj. U slučaju kada se povratkom utvrdi da su i za promenljivu u korenu isprobane sve vrednosti, ustanovljeno je da je problem nezadovoljiv.

Teorema 1 (*Korektnost naivne pretrage sa povratkom.*)**.** *Naivna pretraga sa povratkom se uvek završava. Ako početni skup ograničenja ima rešenje, onda pretraga sa povratkom vraća jedno rešenje, tj. dodelu vrednosti promenljivama. U suprotnom, pretraga sa povratkom vraća indikator da ne postoji rešenje (`false`).*

2.2.1.2 Modifikacije naivnog algoritma

Navedeni algoritam je opšti i obično neefikasan, ali se može unaprediti i prilagoditi raznim tipovima problema koje treba rešiti. Modifikacije algoritma su mnogobrojne, a ovde su kratko navedene samo one najčešće korišćene. Detaljniji pregled

```
function BT (Buduće, Prošle)
  if Buduće = ∅
    return Prošle
  izaberi promenljivu x iz skupa Buduće (x je x_i sa najmanjim indeksom i)
  for each vrednost v iz domena D(x) (počev od najmanje do najveće)
    if Proveri_konzistenciju ({x = v} ∪ Prošle)
      r = BT (Buduće \ {x}, Prošle ∪ {x = v})
      if r ≠ false
        return r
  return false

function Proveri_konzistenciju (Prošle)
  for each c iz C
    if sve promenljive iz c su u Prošle
      if Prošle ne zadovoljavaju c
        return false
```

Slika 2.5: Algoritam naivne pretrage sa povratkom za rešavanje problema CSP. Početni poziv funkcije je: `BT` (`X`, ∅). Skup domena promenljivih i skup ograničenja (`C`) se smatraju globalnim promenljivama.

ovih modifikacija može se pronaći u glavi 4 knjige *Handbook of Constraint Programming* [108]. U toj glavi, autor naglašava da ne postoji verzija pretrage sa povratkom koja je pogodna za sve vrste problema i da se pri poređenju raznih verzija pokazuje da je neka verzija bolja od drugih samo na određenim klasama problema.

Razmatraćemo tri vrste modifikacija naivnog algoritma:

- Modifikacije prvog tipa se odnose na opšte odluke o načinu formiranja stabla, na primer, način za formiranje grana, redosled biranja promenljivih, redosled biranja vrednosti promenljivih, itd. Ovo se postiže korišćenjem pravila koja se zovu *heuristike* i koje određuju kako izabrati između više alternativa.

- Određene modifikacije pretrage se mogu iskoristiti za obradu situacija kada dođe do konflikta, da bi se ubrzala pretraga. Na primer, može se zapamtiti kombinacija vrednosti promenljivih koja dovodi do konflikta, da bi se izbegla ova kombinacija u budućnosti. Stoga, ove modifikacije se mogu smatrati određenim oblikom učenja. Ovde spadaju *pretraga sa skokom unazad* (eng. *backjumping*) i *pretraga sa obeležavanjem* (eng. *backmarking*).

- Naivni algoritam se može modifikovati tako da se koristi u kombinaciji sa *propagiranjem ograničenja*, tehnikom koja se za razliku od prethodno navedenih

upotrebljava za izbegavanje dolaženja do konflikta. Ova tehnika se nekada i samostalno koristi za rešavanje problema CSP ali se mnogo češće koristi u kombinaciji sa pretragom sa povratkom.

Slede opisi prve dve vrste modifikacija, dok je tehnika propagiranja ograničenja zbog svog posebnog značaja izdvojena i posebno opisana u poglavlju 2.2.2.

Strategije grananja. Način na koji se novo dete čvor dodeljuje čvoru stabla zove se *strategija grananja* (eng. *branching strategy*). Strategija koja je korišćena u naivnom algoritmu se zove *nabrajanje* (ima različito značenje od istoimenog termina uvedenog na početku poglavlja 2.2 koji označava metodu rešavanja), tj. svakoj grani odgovara po jedna vrednost promenljive koja se razmatra. Dosta korišćena je i *strategija binarnog izbora* – generišu se dve grane pri čemu jednoj grani odgovara dodela vrednosti promenljivoj, a drugoj odgovara zabrana da promenljiva uzima tu vrednost. Na primer, jednoj grani može odgovarati dodela vrednosti $x = v$ (v je iz domena promenljive x), a drugoj izbacivanje vrednosti v iz domena x ili dodavanje ograničenja $x \neq v$. U slučaju $x \neq v$, ne nastavlja se obavezno sa drugim binarnim izborima po promenljivoj x, već se binarni izbori mogu uraditi i po drugim promenljivama. Kod *strategije podele domena* uvek se generišu dve grane, koje odgovaraju smanjenju domena promenljive. Na primer, jednoj grani može odgovarati smanjenje domena promenljive x na vrednosti manje od $v \in D(x)$, a drugoj smanjenje domena x na vrednosti koje su veće ili jednake od v. Predloženo je i grananje po skupovima u kome svakoj grani odgovara jedan skup vrednosti promenljive a podela u skupove je rađena pomoću algoritma klasterovanja iz oblasti mašinskog učenja [8]. Urađeno je i više poređenja i eksperimentalnih evaluacija različitih strategija grananja ([8, 67]).

Biranje promenljive. Postoji više načina za biranje promenljive po kojoj će se vršiti grananje iz skupa do tada neoznačenih promenljivih. Najčešće korišćeni način je da se uvek bira promenljiva sa najmanjim brojem preostalih vrednosti u domenu. Biranjem promenljivih sa malim brojem vrednosti u domenu se postiže da iz razmatranog čvora izlazi manji broj grana kao i da se za promenljive za koje postoji manje izbora prvo bira vrednost. Drugi popularan način je odabir promenljive koja se javlja u najvećem broju ograničenja. Motivacija za ovakav odabir leži u smanjenju veličine domena drugih promenljivih – u što većem broju ograničenja promenljiva učestvuje, to će (verovatno) veći broj vrednosti biti izbačen iz domena drugih promenljivih (za ovo se koriste tehnike propagiranja ograničenja a više detalja o ovim

tehnikama se nalazi u poglavlju 2.2.2). Zadnjih godina se sve više koriste tehnike koje vrše dodeljivanje težina promenljivama, a prilikom biranja promenljive, uzima se ona sa najvećom težinom. Težina promenljive je zbir težina ograničenja koja su nad njom definisana, gde se težine ograničenja mogu izračunavati na različite načine. Na primer, svaki put kada se propagiranjem ograničenja domen neke promenljive isprazni, uvećava se težina ograničenja koje je uzrokovalo pražnjenje domena [25]. Težine ograničenja određivane su i pomoću metaheuristika [97, 72], tako što se traže dodele vrednosti promenljivama koje minimizuju broj nezadovoljenih ograničenja (metaheuristike su opisane u poglavlju 2.2.3). Za svaku pronađenu dodelu povećava se težina ograničenjima koja su tom dodelom nezadovoljena. Navedeni načini odabira mogu da se vrše bilo dinamički ili statički. Kod *dinamičkog* odabira se u svakom čvoru biranje promenljive iznova sprovodi, na osnovu trenutno dodeljenih vrednosti promenljivama i trenutnog skupa ograničenja. Kod *statičkog* načina odabira, redosled promenljivih je fiksiran pre nego što se pokrene pretraga. Pored navedenih tehnika postoje i mnoge druge, a njihov kraći pregled može se naći u radu koji su napisali Džafari i Mohoub [97].

Biranje vrednosti. Po odabiru promenljive po kojoj je odlučeno da se izvrši grananje potrebno je odrediti i redosled kojim će se dodeljivati vrednosti toj promenljivoj. Većina verzija pretrage sa povratkom obično se odlučuje za vrednost koja će najverovatnije biti deo rešenja; ukoliko ona ne dovede do rešenja bira se sledeća najverovatnija, itd. Odabir se vrši na osnovu procene broja rešenja u kojima će vrednost učestvovati ili procene verovatnoće da vrednost bude deo rešenja. Treba naglasiti da je u pitanju samo procena – ukoliko bi se znalo da će neka vrednost biti deo rešenja, onda bi se i odabrala ta vrednost. Neke od poznatih metoda za brojanje rešenja zasnivaju se na korišćenju relaksacije problema koji se predstavlja stablom ([43]) i na biranju vrednosti čijim se izborom maksimizuje proizvod veličine domena neinicijalizovanih promenljivih ([58]). Biranje vrednosti vršeno je i dodelom težina vrednostima promenljivih ([97]), tako što se težine vrednosti smanjuju kada su te vrednosti deo dodeljenih vrednosti kojima je neko ograničenje postalo nezadovoljeno, a povećavaju u suprotnom.

Pretraga sa skokom unazad. Jedan od glavnih nedostatka naivne pretrage sa povratkom je da se u slučaju kada su isprobane sve vrednosti promenljive u nekom čvoru vrši povratak na nivo u stablu odmah iznad nivoa tog čvora. Algoritam

ne otkriva razloge zbog kojih dolazi do neuspeha, na primer, zbog dve konfliktne vrednosti para promenljivih, pa vrlo brzo može doći do ponovljenog neuspeha iz istog razloga. Ovaj problem se može prevazići pomoću *pretrage sa skokom unazad* [11], koja se nekada naziva i *inteligentna pretraga sa povratkom*. Ako posmatramo stablo pretrage, kada se algoritmom dođe do konfliktnog čvora, povratak se može izvršiti na čvor koji je i nekoliko nivoa iznad konfliktnog čvora, tako što se izračunava koji je čvor uticao na nastanak konflikta. Ilustrovaćemo pretragu sa skokom unazad na primeru.

Tabela 2.1 ilustruje situaciju do koje se došlo prilikom rešavanja nekog problema CSP nakon dodele vrednosti 7 promenljivih (ovaj primer se ne odnosi na konkretan problem već na moguću situaciju do koje se došlo prilikom rešavanja nekog problema). Smatraćemo da je promenljiva x_i dobila svoju vrednost na i-tom nivou, bez ulaženja u detalje koju konkretnu vrednost je ova promenljiva dobila. Promenljiva x_7 ima domen $\{1, 2, 3\}$ ali se dodelom bilo koje od ovih vrednosti toj promenljivoj otkriva da barem jedno ograničenje nije zadovoljeno. Na primer, u slučaju da x_7 dobije vrednost 1 ograničenja $C_{1,3,7}$ i $C_{2,4,7}$ nisu zadovoljena. Za sve promenljive nad kojima je definisano neko od ograničenja koje nije zadovoljeno kažemo da *učestvuju u konfliktu*. U slučaju dodele $x_7 = 1$, u konfliktu učestvuju promenljive x_1 i x_3 (zbog ograničenja $C_{1,3,7}$) i promenljive x_2 i x_4 (zbog ograničenja $C_{2,4,7}$).

Tabela 2.1: Primer situacije u kojoj je došlo do konflikta pri pretrazi. U koloni je za svako ograničenje označeno koje promenljive učestvuju u tom ograničenju. Poslednja vrsta sadrži vrednost čijom je dodelom promenljivoj x_7 došlo do nezadovoljivosti određenih ograničenja.

Nivo (promenljiva)	$C_{1,3,7}$	$C_{2,4,7}$	$C_{1,2,3,7}$	$C_{1,2,7}$	$C_{3,7}$
1 (x_1)	x		x	x	
2 (x_2)		x	x	x	
3 (x_3)	x		x		x
4 (x_4)		x			
5 (x_5)					
6 (x_6)					
7 (x_7)	x	x	x	x	x
vrednost x_7	1		2		3

Pošto je čvor koji odgovara promenljivoj x_7 konfliktan, mora se izvršiti povratak i isprobavanje naredne vrednosti promenljive x_6. Međutim, pošto ova promenljiva

uopšte ne učestvuje u konfliktu, kod novog isprobavanja vrednosti promenljive x_7 se bez obzira na izabranu vrednost za x_6 dešava isti konflikt kao i u prethodnoj situaciji (novi konfliktni čvor). Što je više vrednosti u domenu x_6, to je veća i potrošnja vremena na ovaj deo prostora pretrage koji svakako ne dovodi do rešenja, a uvek dovodi do istih nezadovoljenih ograničenja. Situacija je dodatno pogoršana činjenicom da ni promenljiva x_5 ne učestvuje u konfliktu – po isprobavanju svih vrednosti promenljive x_6 vraćanje se vrši još jedan nivo iznad i isprobava nova vrednost za x_5, pa se proces pretrage i nastanka konflikta iznova puno puta ponavlja iako ni x_5 ni x_6 nemaju nikakav uticaj na konflikt.

Rešenje problema je popeti se za više nivoa (*skočiti unazad*), tj. vratiti se završetkom rekurzivnih poziva do neke promenljive koja učestvuje u konfliktu i probati sa novom vrednošću te promenljive. Poželjno je da skok bude što je moguće veći, da bi se preskočio što veći prstor pretrage, ali je ključno da se skokom unazad ne sme preskočiti čvor u čijem podstablu se možda nalazi rešenje problema. Najjednostavniji način za određivanje promenljive kojoj treba promeniti vrednost je da se izabere promenljiva koja učestvuje u konfliktu a nalazi se na najvećem mogućem nivou (u primeru se vraćanje vrši na nivo 4). Međutim, promenom vrednosti promenljive x_4 može postati zadovoljeno ograničenje $C_{2,4,7}$, ali sigurno neće biti zadovoljeno ograničenje $C_{1,3,7}$. Bolja ušteda se postiže sledećim postupkom: za svako ograničenje ($C_{1,3,7}$ i $C_{2,4,7}$ u primeru) koje nije zadovoljeno zbog neke vrednosti ($x_7 = 1$) odrede se promenljive različite od konfliktne (x_7), a maksimalnih nivoa u tim ograničenjima (x_3 i x_4) i među njima odredi ona minimalnog nivoa (x_3). Ponavljanjem ovog postupka se za sve vrednosti konfliktne promenljive (1, 2, 3) pronađu minimalni nivoi (3, 2, 3), potom se određuje maksimalni među njima i skok unazad se vrši na taj nivo (3).

Bitno je napomenuti da u pogledu vremena, određivanje nivoa na koji treba skočiti nije jeftina operacija. Postoje situacije u kojima se više vremena potroši na određivanje nivoa nego što bi bilo potrošeno na pretragu. Stoga se uprkos tome što je poželjnije da skok bude što je moguće veći, često koriste algoritmi kojima se postiže manji skok, ali se za određivanje ovog skoka troši mala količina vremena.

Pretraga sa obeležavanjem. Iako pretraga sa skokom unazad obično značajno poboljšava efikasnost naivne pretrage, i ona ima svoje nedostatke. Jedan od nedostataka je i trošenje vremena na proveru konzistencije za skup vrednosti promenljivih za koji se već ranije utvrdilo da ne zadovoljava postojeća ograničenja. Ovaj nedostatak

se prevazilazi pomoću modifikacije pretrage koja se zove *pretraga sa obeležavanjem* [11, 42].

Pretraga sa obeležavanjem je *tehnika učenja* koja koristi dodatan memorijski prostor da bi se prikupile informacije u dva slučaja (naglasimo da je podrazumevano da je redosled dodeljivanja vrednosti promenljivama fiksiran na $x_1, \ldots, x_n$, što je slučaj i kod naivne pretrage predstavljene na slici 2.5). Ilustrovaćemo pretragu sa obeležavanjem na primeru.

Razmotrimo već pomenuti primer u tabeli 2.1 i pokušaj dodele vrednosti 1 promenljivoj x_7. Situacija koja je prikazana u toj tabeli se tokom pretrage može ponavljati više puta. Stoga je moguće zapamtiti tu situaciju kako bi u budućnosti bilo utrošeno znatno manje vremena za određivanje promenljive kojoj treba promeniti vrednost. Za promenljivu x_7 i vrednost 7 pamti se najmanji mogući indeks j, takav da je dodela vrednosti $x_1, \ldots, x_j, x_7$ nekonzistentna, tj. takav da postoji neko nezadovoljeno ograničenje nad podskupom tih promenljivih (u primeru je $j = 3$). Takođe, pamti se i minimalni indeks $k < 7$ promenljive koja je promenila vrednost od poslednje dodele vrednosti promenljivoj x_7 (u primeru je $k = 6$). Indeksi j i k se ažuriraju kroz proces pretrage. Ako u nekoj situaciji ponovo bude razmatrana mogućnost da se izvrši dodela $x_7 = 1$, onda je u slučaju $j < k$ $(3 < k)$ nepotrebno proveravati tu dodelu i prelazi se na razmatranje sledeće vrednosti promenljive x_7. Ukoliko je $j \geq k$, onda se mora krenuti sa detaljnom proverom konzistencije za $x_7 = 1$.

Preciznije, opis razmatranja u pretrazi sa označavanjem se može opisati sledećim postupkom. Prvo, za svaku promenljivu x_i i vrednost a iz njenog domena pamte se određene informacije o situaciji kada je x_i poslednji put dobila vrednost a. Konkretno, pamti se minimalni indeks $j < i$ takav da je dodela vrednosti $x_1, \ldots, x_j, x_i$ nekonzistentna, odnosno 0 ako je za svaki indeks $j < i$ ta dodela vrednosti konzistentna. Ovaj indeks se ažurira svaki put kada x_i dobije vrednost a proverom konzistencije za dodeljene vrednosti, i to sledećim redosledom: prvo za x_1, x_i, potom ako je potrebno za x_1, x_2, x_i, potom ako je potrebno za x_1, x_2, x_3, x_i, itd. Drugo, za svaku promenljivu x_i čuva se minimalni indeks $k < i$ promenljive koja je promenila vrednost od poslednje dodele vrednosti promenljivoj x_i. To se postiže tako što se pri dodeli vrednosti bilo kojoj promenljivoj x_l za sve promenljive sa indeksom $i > l$ ažurira minimalni indeks za x_i da bude `min` (k, l), gde je k prethodno zapamćeni minimalni indeks za x_i.

Prikupljene informacije se koriste da bi se izbegle provere konzistencije. Svaki

put kada se promenljivoj x_i dodeli vrednost a, vrši se poređenje dva zapamćena indeksa j i k, čije je značenje opisano u prethodnom pasusu. Ukoliko je $j < k$ onda je dodela vrednosti promenljivama $x_1, \ldots, x_j, x_i$ i dalje nekonzistentna kao i pre, pošto nijedna od promenljivih $x_1, \ldots, x_j$ nije promenila svoju vrednost. Ukoliko je $j \geq k$ onda je potrebno proveriti konzistenciju za dodeljene vrednosti nekih nizova promenljivih, ali ne svih – nema potrebe proveravati konzistencije za dodeljene vrednosti kombinacija promenljivih $x_1, x_i, \quad x_1, x_2, x_i, \quad \ldots \quad , x_1, \ldots, x_{k-1}, x_i$. Dakle, u nekim situacijama se u potpunosti preskače provera konzistencije dok se u drugim smanjuje broj provera konzistencije, pa se ovom pretragom može značajno uštedeti na vremenu, ali po cenu zauzeća dodatnog memorijskog prostora.

Odnos pretrage sa skokom unazad i pretrage sa obeležavanjem. Obe pomenute tehnike se mogu koristiti u okviru pretrage sa povratkom. Razmotrimo ponovo primer u tabeli 2.1. Kada se prvi put pretraga nađe u stanju predstavljenim tim primerom, tehnika pretrage sa skokom unazad se koristi da se odredi kojoj promenljivoj je potrebno promeniti vrednost. Pri tom razmatranju se za promenljivu x_7 i svaku od vrednosti 1, 2 i 3 određuje pomenuti indeks j iz pretrage sa obeležavanjem. Dakle, indeks j se već izračunava u pretrazi sa skokom unazad, a potrebno je zapamtiti ga, dok se indeks k mora i izračunati i zapamtiti. Ušteda primenom pretrage se obeležavanjem se postiže tek kasnije, kada se ponovo pokuša sa dodelom vrednosti 1, 2 ili 3 promenljivoj x_7.

2.2.2 Algoritmi propagiranja ograničenja

Jedan bitan nedostatak naivne pretrage sa povratkom je da je prostor pretrage izuzetno veliki, čak i u slučajevima kada se on može značajno smanjiti. Naime, u funkciji `Proveri_konzistenciju` prikaznoj na slici 2.5 se ne postiže smanjivanje domena promenljivih. Ovo se može prevazići pomoću algoritama za propagiranje ograničenja, kojima je posvećena velika pažnja i veoma veliki broj radova, pa ih ovde detaljno opisujemo.

Cilj algoritama iz ove grupe ([4, 108]) je da problem CSP prevedu u lakši problem sa istim skupom rešenja kao polazni problem, gde termin 'lakši' obično označava da je prostor pretrage (broj n-torki Dekartovog proizvoda $D_1 \times \ldots \times D_n$) smanjen. To se postiže tako što se odstranjuju vrednosti iz domena promenljivih koje ne mogu biti deo rešenja i tako što se ojačavaju postojeća ili dodaju nova ograničenja. Ako domen neke promenljive postane prazan, onda ne postoji rešenje zadatog problema.

Korišćenje propagiranja ograničenja obično nije dovoljno da se reši problem, ali se u kombinaciji sa drugim tehnikama rešavanja pokazuje kao vrlo efikasan način za rešavanje problema CSP. U algoritmima prikazanim na slikama 2.5 i 2.7 se propagiranje ograničenja može dodati u funkciju `Proveri_konzistenciju`.

Nastanak pojma propagiranja ograničenja je tesno povezan sa predstavljanjem problema CSP pomoću *grafa ograničenja*. Naime, dokazano je da se svaki problem CSP može transformisati u njemu ekvivalentan problem (problem sa istim skupom rešenja), ali koji sadrži samo unarna i binarna ograničenja. Ova transformacija je često veoma računski zahtevna i neretko dovodi do ogromnog povećanja broja ograničenja, pa se stoga retko koristi. Pogodnost transformacije u navedeni oblik je u tome što se problem CSP može predstaviti grafom ograničenja u kome čvorovi predstavljaju promenljive sa pridruženim unarnim ograničenjima, a lukovi binarna ograničenja. Rešavanje problema CSP korišćenjem grafa ograničenja se sastoji od iterativnih transformacija problema u sve jednostavnije ekvivalentne probleme kojima se pridružuju odgovarajući grafovi ograničenja. Transformacija se vrši izbacivanjem grana, tj. uvažavanjem binarnih ograničenja problema.

Propagiranjem ograničenja se problem CSP najčešće prevodi u problem koji zadovoljava određene uslove koji se zovu uslovi *lokalne konzistencije*. Postoji više vrsta lokalne konzistencije (neke su opisane u poglavljima koja slede) i svaki problem CSP se može prevesti u problem sa istim skupom rešenja koji zadovoljava uslove proizvoljne lokalne konzistencije primenom precizno utvrđenog postupka. Ovo je vrlo različito od pojma konzistencije problema, jer problem može biti ili konzistentan ili nekonzistentan i prevođenjem u druge probleme sa istim skupom rešenja konzistencija problema se ne menja. Da bi se posebno istakla razlika između lokalne konzistencije i konzistencije, druga se ponekad zove *globalna konzistencija*. Postoji veliki broj tehnika propagiranja ograničenja, ali primenjivanje većeg broja ovih tehnika može biti vremenski veoma zahtevno. Zato se obično primenjuju samo efikasne tehnike sa ciljem da se prostor pretrage smanji do određenog stepena.

U nastavku slede opisi nekoliko tehnika propagiranja ograničenja i uslova koje problemi CSP zadovoljavaju kada se na njih primene ove tehnike. Kao što je već napomenuto nakon definicije 1, C_i će označavati ograničenje definisano na promenljivoj x_i, $C_{i,j}$ ograničenje definisano na promenljivama x_i i x_j, itd.

U prvim delovima naredna tri poglavlja, prvo opisujemo tri različite vrste lokalne konzistencije i operacije koje je potrebno iterativno primenjivati da bi se problem

preveo u oblik koji je u odgovarajućem stanju lokalne konzistencije. Primenom operacija se u novom obliku problema zadržavaju sva (i samo ona) rešenja koja su bila i rešenja početnog problema. Potom su u svakom od tih poglavlja kratko opisani i algoritmi koji iterativno primenjuju operacije da bi preveli problem u oblik koji je u odgovarajućem stanju lokalne konzistencije. U pseudokodovima prikazanim na slikama 2.5 i 2.7 ovi algoritmi se dodaju u funkciju `Proveri_konzistenciju`, a razlikuju se po podskupovima promenljivih koje razmatraju. U zavisnosti od podskupa koji se razmatra, dobijaju se različite vrste pretraga sa povratkom, od kojih svaka ima svoje ime (ove pretrage su tema poglavlja 2.2.2.5, 2.2.2.6 i 2.2.2.7). Dakle, tokom pretrage sa povratkom se problem može veliki broj puta prevoditi u oblik koji je u odgovarajućem stanju lokalne konzistencije.

2.2.2.1 Čvorna konzistencija

Definicija 4. *Problem CSP je u stanju* čvorne konzistencije *ili* konzistencije čvora *(eng.* node consistency*) ako je za svako unarno ograničenje C_i ispunjeno da za svaku vrednost u iz domena D_i, važi da $u \in C_i$.*

Dovođenje problema CSP u stanje čvorne konzistencije postiže se smanjivanjem domena promenljivih koristeći samo unarna ograničenja do trenutka kada nijedno takvo smanjivanje ne može da se izvrši (smanjivanje je potrebno obaviti najviše jednom za svaku promenljivu, pošto ono ne utiče na domene drugih promenljivih). Smanjivanje podrazumeva da se iz domena promenljive nad kojom je definisano unarno ograničenje uklanjaju vrednosti koje ne zadovoljavaju to ograničenje, tj. za neku promenljivu x_i i unarno ograničenje C_i smanjuje se domen te promenljive izvršavanjem operacije:

$$D_i := D_i \cap C_i$$

Primer 6. *Ako problem CSP sadrži celobrojnu promenljivu x_i sa domenom $D_i = \{1, \ldots, 10\}$ i ograničenje $C_i = \{1, \ldots, 5\}$ (odgovara uslovima $x_i \geq 1$ i $x_i \leq 5$) onda se domen D_i smanjuje pomoću operacija $D_i := D_i \cap C_i = \{1, \ldots, 5\}$.*

Posmatranjem navedene operacije i prethodnog primera moglo bi se postaviti pitanje da li je umesto navedene operacije dovoljno koristiti operaciju $D_i := C_i$? To nije dovoljno, jer se algoritmima za postizanje drugih vrsta konzistencije (npr. lučne konzistencije) mogu izbaciti neke vrednosti iz domena D_i, i time dovesti problem CSP u stanje u kome sve vrednosti domena D_i nisu istovremeno i u ograničenju C_i.

Algoritam za postizanje čvorne konzistencije je vrlo jednostavan: prolazi se redom kroz sva unarna ograničenja i primenom odgovarajuće operacije smanjuje domen promenljive na koju se odnosi ograničenje.

2.2.2.2 Lučna konzistencija

Definicija 5. *Problem CSP je u stanju* lučne konzistencije *ili* konzistencije luka *(eng.* arc consistency*) [17] ako je za svako binarno ograničenje $C_{i,j}$ ispunjeno da za svaku vrednost u iz domena D_i postoji vrednost v iz domena D_j tako da $(u, v) \in C_{i,j}$. Simetrično, za svaku vrednost v iz domena D_j mora postojati vrednost u iz domena D_i tako da $(u, v) \in C_{i,j}$.*

Dovođenje problema CSP u stanje lučne konzistencije postiže se smanjivanjem domena promenljivih koristeći samo binarna ograničenja do trenutka kada nijedno takvo smanjivanje ne može da se izvrši (smanjivanje se može obaviti više puta iznova za neki par promenljivih (x_i, x_j), pošto smanjivanje domena drugih parova može uticati na novo smanjivanje domena para (x_i, x_j)). Smanjivanje podrazumeva da se iz domena promenljivih uklanjaju vrednosti koje ne zadovoljavaju binarno ograničenje, tj. za neke promenljive x_i i x_j i binarno ograničenje $C_{i,j}$ smanjuju se domeni tih promenljivih izršavanjem operacija:

$$D_i := \{u \in D_i | (\exists v \in D_j)(u, v) \in C_{i,j}\}$$

$$D_j := \{v \in D_j | (\exists u \in D_i)(u, v) \in C_{i,j}\}$$

Primetimo da navedene operacije ne uzimaju u obzir redosled promenljivih u paru, tj. smanjivanje domena koje odgovara paru (x_i, x_j) je isto kao i smanjivanje domena koje odgovara paru (x_j, x_i).

Primer 7. *Neka problem CSP sadrži celobrojne promenljive x_i, x_j i x_k sa domenima $D_i = D_j = D_k = \{1, 2, 3\}$ i ograničenja $C_{i,j} = \{(1, 2), (1, 3), (2, 3)\}$, i $C_{j,k} = \{(1, 2), (1, 3), (2, 3)\}$ (odgovaraju redom uslovima $x_i < x_j$ i $x_j < x_k$). Iz domena D_i i D_j se uklanjaju vrednosti koje nisu saglasne sa ograničenjem $C_{i,j}$, tj. ovi domeni postaju $D_i = \{1, 2\}$ i $D_j = \{2, 3\}$ (na primer, iz domena D_i je izbačena vrednost 3, jer ne postoji vrednost v koja pripada domenu D_j, takva da $(3, v) \in C_{i,j}$). Iz domena D_j i D_k se uklanjaju vrednosti koje nisu saglasne sa ograničenjem $C_{j,k}$, tj. ovi domeni postaju $D_j = \{2\}$ i $D_k = \{3\}$. Posmatrajući ponovo ograničenje $C_{i,j}$ uočavamo da iz domena D_i i D_j treba ukloniti vrednosti koje nisu saglasne sa*

ograničenjem $C_{i,j}$, tj. domen D_i se smanjuje i konačno domeni postaju $D_i = \{1\}$, $D_j = \{2\}$ i $D_k = \{3\}$.

Primer 8. *Ako problem CSP sadrži celobrojne promenljive x_i i x_j sa domenima $D_i = D_j = \{1, 2, 3\}$ i ograničenje $C_{i,j} = \{(1, 1), (2, 2), (3, 3)\}$ (odgovara uslovu $x_i = x_j$) onda iz domena D_i i D_j ne treba ukloniti nijednu vrednosti razmatrajući ograničenje $C_{i,j}$, jer za svaku vrednost promenljive x_i (odnosno x_j) postoji ista ta vrednost u domenu D_j (D_i).*

Prikazane operacije za postizanje lučne konzistecije su najšire korišćene operacije za propagiranje ograničenja. Postoji više algoritama koji na različit način iterativno upotrebljavaju ove operacije da bi problem preveli u lakši problem. Najjednostavniji je AC-1 [84] i u tom algoritmu se u svakom ciklusu prolazi kroz sve moguće parove promenljivih i smanjuju domeni tih promenljivih korišćenjem navedenih operacija. Algoritam se završava kada u nekom ciklusu ni kod jedne promenljive ne dođe do smanjenja domena.

Napredniji algoritam za postizanje lučne konzistencije je AC-3 [84] i on se takođe sastoji iz ciklusa (postoji i algoritam AC-2 ali je on samo specijalan slučaj algoritma AC-3). Svakom ciklusu se pridružuje lista parova promenljivih koji se obrađuju u tom ciklusu, pri čemu se inicijalno u prvom ciklusu nalaze svi parovi promenljivih, a ostali ciklusi su prazni. Ako je u nekom ciklusu obradom ograničenja $C_{i,j}$ izmenjen domen jedne od promenljivih na kojima je ono definisano, na primer domen promenljive x_i, onda se u listu parova promenljivih koji će biti obrađeni u narednom ciklusu dodaju svi parovi promenljivih koji uključuju x_i, ali bez para (x_i, x_j). Dakle, za razliku od algoritma AC-1, u ciklusu se razmatraju samo parovi promenljivih gde je domen barem jedne od njih menjan u prethodnom ciklusu. Na taj način se obično u svakom ciklusu razmatra znatno manji broja parova promenljivih što često dovodi do značajnih poboljšanja u efikasnosti.

2.2.2.3 Putna konzistencija

Za razliku od prethodne dve vrste konzistencije koje su se odnosile na smanjivanje domena promenljivih, kod putne konzistencije razmatramo pojačavanje ograničenja (ograničenja postaju strožija, tj. dozvoljavaju manji broj kombinacija vrednosti).

Definicija 6. *Pretpostavimo da je za svaki par promenljivih definisano ograničenje na te dve promenljive (ako ograničenje ne postoji uvodi se Dekartov proizvod domena*

ovih promenljivih kao ograničenje). Problem CSP je u stanju putne konzistencije *(eng.* path consistency*) ako je za svake tri promenljive x_i, x_j i x_k ispunjeno da za svaki par vrednosti $(u, v) \in C_{i,j}$ postoji vrednost $w \in D_k$ tako da $(u, w) \in C_{i,k}$ i $(w, v) \in C_{k,j}$.*

Dovođenje problema CSP u ovo stanje postiže se pojačavanjem binarnih ograničenja na promenljivama do trenutka kada nijedno takvo pojačavanje ne može da se izvrši. Ako vrednosti u i v iz domena promenljivih x_i i x_j zadovoljavaju ograničenje $C_{i,j}$, a ne postoji vrednost w iz domena promenljive x_k takva da važe oba iskaza $(u, w) \in C_{i,k}$ i $(w, v) \in C_{k,j}$, onda se par (u, v) može izbaciti iz ograničenja $C_{i,j}$. Dakle, do postizanja putne konzistencije se dolazi primenom operacije:

$$C_{i,j} := \{(u, v) \in C_{i,j} | (\exists w \in D_k)(u, w) \in C_{i,k} \wedge (w, v) \in C_{k,j}\}$$

Analogno se vrši pojačavanje za ograničenja $C_{i,k}$ i $C_{j,k}$, pri čemu se razmatraju sve moguće trojke promenljivih (x_i, x_j, x_k).

Primer 9. *Neka problem CSP sadrži celobrojne promenljive x_i, x_j i x_k sa domenima $D_i = D_j = D_k = \{1, \ldots, 3\}$ i ograničenja $C_{i,j} = \{(1, 1), (2, 2), (3, 3)\}$ (odgovara uslovu $x_i = x_j$), $C_{i,k} = \{(2, 1), (3, 1), (3, 2)\}$ (odgovara uslovu $x_i > x_k$), $C_{k,j} = \{(2, 1), (3, 1), (3, 2)\}$ (odgovara uslovu $x_k > x_j$). Ako posmatramo promenljive x_i, x_j i x_k i ograničnje $C_{i,j}$, onda se pomoću opisanih operacija ovaj skup smanjuje da bude prazan – na primer, za par $(u, v) = (2, 2)$ postoji samo jedna vrednost $w = 1$, tako da $(2, 1) \in C_{i,k}$ ali ne važi $(1, 2) \in C_{k,j}$. Zato se par $(2, 2)$ uklanja iz skupa $C_{i,j}$. Na sličan način se uklanjaju i ostala dva para. Prazno ograničenje označava da je problem nekonzistentan, što odgovara činjenici da ne postoje vrednosti promenljivih koje bi ispunjavale zadate uslove $x_i > x_k > x_j, x_i = x_j$.*

Kao i u slučaju lučne konzistencije, jednostavnim algoritmom PC-1 ([84, 96]) se u svakom ciklusu prolazi kroz sve trojke promenljivih i vrši pojačavanje ograničenja. Kada u nekom ciklusu ne dođe ni do jednog smanjenja, problem CSP je doveden u stanje putne konzistencije. Napredniji algoritam PC-2 [84] razmatra u svakom novom ciklusu samo ona ograničenja na koja su mogla da utiču smanjenja u prethodnom ciklusu. Može se desiti da za neki par promenljivih nije postojalo ograničenje (odgovaralo je Dekartovom proizvodu domena ovih promenljivih), ali su algoritmom za postizanje putne konzistencije neki parovi vrednosti tih promenljivih eliminisani. Tada se skupu ograničenja dodaje ograničenje za taj par promenljivih i ono onemogućava da te promenljive dobiju eliminisane vrednosti.

2.2.2.4 Međusobni odnosi različitih vrsta konzistencija

Postoje i druge vrste lokalnih konzistencija ali su navedene tri vrste najčešće korišćene. Primena algoritma za postizanje jedne vrste lokalne konzistencije može uticati da problem više ne bude u stanju neke druge vrse lokalne konzistencije, pa se ovi algoritmi mogu naizmenično primenjivati.

Ne postoji direktna povezanost globalne konzistencije i lokalne konzistencije, tj. problem koji je u stanju neke lokalne konzistencije može ali ne mora imati rešenja. Za ilustraciju ovoga dovoljno je pogledati jednostavan primer.

Primer 10. *Neka problem CSP sadrži celobrojne promenljive x_1, x_2 i x_3 sa domenima $D_1 = D_2 = D_3 = \{1, 2\}$ i ograničenja $C_{1,2} = \{(1, 2), (2, 1)\}$, $C_{1,3} = \{(1, 2), (2, 1)\}$, $C_{3,2} = \{(1, 2), (2, 1)\}$ (odgovaraju redom uslovima $x_1 \neq x_2$, $x_1 \neq x_3$ i $x_3 \neq x_2$). Ovaj problem je u stanju čvorne i lučne konzistencije ali nije konzistentan, tj. nema rešenja. Ovaj problem nije u stanju putne konzistencije (na primer, ovo se može utvrditi jer za par $(1, 2) \in C_{1,2}$ ne postoji vrednost $v_3 \in D_3$ takva da $(1, v_3) \in C_{1,3}$ i $(v_3, 2) \in C_{3,2}$).*

Kao što je već rečeno, algoritam pretrage se može modifikovati tako da se sprovode neka od navedenih operacija za postizanje lokalne konzistencije. U algoritmima predstavljenim na slikama 2.5 i 2.7, funkcija `Proveri_konzistenciju` se proširuje da iterativno primenjuje opisane operacije za postizanje neke određene ili više različitih vrsta lokalne konzistencije (navedeni su algoritmi AC-1, AC-2, AC-3, PC-1, PC-2 koji iterativno primenjuju neke od opisanih operacija). Time se sprečava nastanak konflikta, pa se može dodatno zaobići značajan prostor pretrage. Ovde ćemo predstaviti 3 najznačajnija algoritma koji predstavljaju modifikacije pretrage sa povratkom, pri čemu se modifikacije vrše izmenama pomenute funkcije `Proveri_konzistenciju`. Prilikom modifikacije, ta funkcija zadržava svu prethodnu funkcionalnost i proširuje se da obavlja dodatne zadatke.

Slede opisi 3 modifikacije pretrage sa povratkom. Ovi algoritmi se razlikuju *u skupu parova promenljivih (tj. binarnih ograničenja definisanih na parovima) koje razmatraju prilikom propagiranja ograničenja.* Za propagiranje ograničenja koristi se neki od već opisanih algoritama za postizanje lučne konzistencije (npr. AC-1 ili AC-3), koji koriste već opisane operacije u delu o lučnoj konzistenciji.

2.2.2.5 Proveravanje unapred

Proveravanje unapred (eng. *forward checking*) je jednostavan algoritam za sprečavanje budućih konflikta. Pri pokušaju dodele vrednosti v tekućoj promenljivoj x iz domena svih budućih promenljivih (privremeno) se izbacuju sve vrednosti čijom bi dodelom bilo nezadovoljeno neko od binarnih ograničenja nad tekućom promenljivom. Preciznije, primenjuju se operacije za dovođenje problema u stanje lučne konzistencije ali samo na binarna ograničenja između tekuće promenljive i promenljivih koje nisu dobile vrednost. Ukoliko se primenom operacija dobije da domen neke buduće promenljive postaje prazan, onda je utvrđeno da se trenutno parcijalno rešenje ne može proširiti do rešenja problema dodeljivanjem vrednosti v promenljivoj x i pokušava se sa narednom vrednošću za x (pri tome se domeni budućih promenjivih vraćaju da budu jednaki domenima pre dodele vrednosti v promenljivoj x). Primetimo da su u trenutku pokušaja dodeljivanja vrednosti tekućoj promenljivoj sve vrednosti domena te promenljive konzistentne sa prošlim promenljivama, pa provera zadovoljenosti ograničenja zadatih nad tim promenljivama i novom promenljivom nije potrebna.

2.2.2.6 Delimičan pogled unapred

Delimičan pogled unapred (eng. *partial look ahead*) je algoritam koji omogućava uklanjanje više vrednosti iz domena od proveravanja unapred, ali je i računski zahtevniji. Dodatna prepostavka je da postoji uređenje među promenljivama, tj. promenljive dobijaju vrednosti u redosledu $x_1, \ldots, x_n$. U ovom algoritmu razmatraju se i binarna ograničenja između budućih promenljivih a ne samo binarna ograničenja između tekuće i budućih promenljivih. Pri pokušaju dodele vrednosti v tekućoj promenljivoj x, iterativno se primenjuju operacije za dovođenje problema u stanje lučne konzistencije na pomenutim ograničenjima. Pri tome se operacije na budućim promenljivama primenjuju tako da se razmatranjem ograničenja $C_{i,j}$ $(i < j)$ uklanjaju samo vrednosti iz domena promenljive sa većim indeksom, tj. samo vrednosti iz domena promenljive x_j. Ukoliko se primenom operacija dobije da domen neke buduće promenljive postaje prazan, onda je utvrđeno da se trenutno parcijalno rešenje ne može proširiti do rešenja problema dodeljivanjem vrednosti v promeljivoj x i pokušava se sa narednom vrednošću za x.

2.2.2.7 Održavanje lučne konzistencije

Održavanje lučne konzistencije (eng. *Maintaining arc consistency*, obično se koristi skraćenica MAC), je algoritam koji proverava sve parove promenljivih. Naziva se i *potpuni pogled unapred* (eng. *full look ahead*). Ovim algoritmom se stalno održava lučna konzistencija na celokupnom skupu promenljivih. Razlika u odnosu na delimični pogled unapred je da se ovde ne uvodi uređenje promenljivih (u primeru u prehodnom pasusu za ograničenje $C_{i,j}$ $(i < j)$ ne bi bile uklanjane samo vrednosti iz domena promenljive x_j, već i vrednosti iz domena promenljive x_i). Dakle, primenjuju se operacije za dovođenje problema u stanje lučne konzistencije na binarna ograničenja između tekuće promenljive i promenljivih koje nisu dobile vrednost, kao i na binarna ograničenja između promenljivih koje nisu dobile vrednost.

Može se steći utisak da je od tri predstavljene pretrage MAC algoritam najbolji, jer se njegovom primenom dolazi do najmanjih domena promenljivih. Međutim, za primenu ove pretrage je potrebno i najviše vremena. U praksi je potrebno proceniti koja od tehnika će doprineti dovoljnom smanjivanju domena za neko prihvatljivo vreme (ovo se može utvrditi i empirijski, tj. isprobavanjem različitih pretraga). Stoga se svaka od tri pretrage koristi u određenim situacijama, kada se proceni da ju je pogodno upotrebiti.

2.2.3 Lokalna pretraga

Uprkos ogromnom napretku u razvoju potpunih algoritama za rešavanje problema CSP, *lokalna pretraga* (eng. *local search*) u pojedinim slučajevima jeste jedina metoda za rešavanje nekih kompleksnih problema. Algoritmi lokalne pretrage su po svojoj prirodi pogodni za obradu kriterijuma optimalnosti koji se javljaju u mnogim praktičnim primenama. Opis i pseudokod koji slede se velikim delom zasnivaju na odgovarajućoj glavi o lokalnoj pretrazi u knjizi "Handbook of Constraint Programming" [108].

Glavna ideja kod lokalne pretrage da se prvo odredi (obično nekom heurističkom metodom a ređe na slučajan način) neka početna dodela vrednosti promenljivama u datom problemu, pri čemu dodela može biti nekompletna, ne mora biti optimalna ili čak i ne zadovoljava sva ograničenja problema, i da se iterativno ta dodela poboljšava pomoću manjih promena. Većina algoritama lokalne pretrage se može primeniti i na probleme CSP i na probleme COP (uz neke modifikacije), pa se naredni tekst velikim delom odnosi na rešavanje obe vrste problema, ukoliko nije drugačije na-

glašeno. Postoje različiti tipovi lokalnih pretraga. Najveći broj ovih algoritama koristi i donošenje odluka na slučajan način da bi osigurali da pretraga ne stagnira sa nezadovoljavajućim kandidatom za rešenje i zato se ovi algoritmi zovu *Metode stohastičke lokalne pretrage* (eng. *Stochastic local search methods*). Najpoznatije metode iz ove grupe su metoda promenljivih okolina, slučajno iterativno poboljšavanje, evolutivni algoritmi, simulirano kaljenje, tabu pretraga, dinamička lokalna pretraga, optimizacija kolonijama mrava, optimizacija kolonijama pčela, itd.

Prethodno pomenuta klasa algoritama lokalne pretrage je široko poznata kao *metaheuristike* (eng. *metaheuristics*). Postoji više različitih definicija metaheuristika koje se mogu pronaći u literaturi ([23]). Po jednoj, metaheuristika je iterativni proces koji vodi i modifikuje operacije podređenih heuristika sa ciljem da se dobiju rešenja visokog kvaliteta. U svakoj iteraciji, može se raditi na kompletnim ili nekompletnim rešenjima, kao i na skupu rešenja. Po drugoj definiciji, metaheuristike predstavljaju metode koje optimizuju problem iterativnim poboljšavanjem potencijalnog rešenja uzimajući u obzir zadatu meru kvaliteta. Po trećoj, metaheuristike su procedure koje su dizajnirane da se pronađe, napravi ili odabere neka heuristika za rešavanje nekog problema.

Većina metaheuristika su konceptualno jednostavne i lake za implementaciju. U isto vreme one često pokazuju odlične performanse, fleksibilne su i mogu lako da budu prilagođene promenama u specifikaciji problema. Zbog navedenih razloga, metaheuristike su veoma popularan izbor za rešavanje kompleksnih problema koji nisu u potpunosti jasno definisani u startu.

Predstavićemo jednu od najjednostavnijih metoda lokalne pretrage, koja je puno puta korišćena i prilagođavana.

2.2.3.1 Iterativno poboljšavanje

Iterativno poboljšavanje (eng. *iterative improvement*) se ponekad zove i *penjanje-uz-brdo* (eng. *hill climbing*). Metoda kreće od proizvoljne, heuristički ili slučajno određene dodele vrednosti promenljivama i u svakom koraku se pokušava sa promenom dodele. Ako se promenom dobija bolja dodela (na primer, manje nezadovoljenih ograničenja u slučaju problema CSP ili bolje rešenje u slučaju problema COP), onda se postupak nastavlja sa promenjenom dodelom, u suprotnom sa nepromenjenom. Sa promenama se pokušava sve do trenutka kada se dodela ne može više unaprediti. Proces poboljšanja rešenja kod metoda lokalne pretrage se zove *intenzifikacija*. U slučaju problema COP ova metoda je dobra za nalaženje lokalnog optimuma ali

ne garantuje pronalaženje globalnog optimuma dok u slučaju problema CSP ova metoda ne garantuje pronalaženje rešenja. Kao pokušaj (ne nužno uspešan) da se ovaj problem prevaziđe, koristi se ponovno startovanje od neke druge dodele. Ovaj proces se zove *ekstenzifikacija*.

Najjednostavniji i dosta korišćen algoritam za iterativno poboljšavanje se zove *heuristika minimizacije konflikta* (eng. *min-conflict heuristic*), skraćeno MCH [108]. Ovaj algoritam se koristi za rešavanje problema CSP, ne garantuje pronalaženje rešenja, a njegov pseudokod je prikazan na slici 2.6. Kreće se od početne slučajne dodele vrednosti promenljivama. U svakoj iteraciji se proverava da li trenutna dodela zadovoljava sva ograničenja i ukoliko to jeste slučaj, pronađeno je rešenje problema. U suprotnom se na slučajan način bira promenljiva x koja učestvuje u nekom *konfliktnom skupu K(a)* – skupu svih promenljivih koje se pojavljuju u ograničenjima koja nisu zadovoljena trenutnom dodelom vrednosti a. Potom se određuje vrednost v iz domena promenljive x takva da se dodelom $x = v$ minimizuje broj nezadovoljenih ograničenja. U slučaju više vrednosti sa ovakvim svojstvom na slučajan način se bira jedna među njima. Pretraga se prekida ili u slučaju pronalaska rešenja ili u slučaju dostizanja broja iteracija `Broj_iteracija`. Ponovno startovanje se lako dodaje ovom algoritmu sa ciljem izbegavanja već spomenute mogućnosti zaglavljivanja u lokalnom optimumu.

function MCH (`Broj_iteracija`)
  a = slučajno dodeljivanje vrednosti skupu `X`
  for `brojač` = 1 **to** `Broj_iteracija`
    if a zadovoljava sva `C`
      return a
    x = slučajno izaberi promenljivu iz konfliktnog skupa K(a)
    v = izaberi v iz domena $D(x)$ tako da se dodelom $x = v$ minimizuje
      broj nezadovoljenih ograničenja
    a = a sa promenom $x = v$
  return `false`

Slika 2.6: Heuristika minimizacije konflikta. Skup promenljivih, njihovih domena i ograničenja (`C`) se smatraju globalnim promenljivama.

2.2.4 Postojeći sistemi za rešavanje problema CSP

Postoji veliki broj sistema za rešavanje problema CSP (najčešće ovi sistemi rešavaju i probleme COP). Ovde je fokus na najpoznatijim sistemima koji su ili u široj upotrebi, ili su redovni učesnici takmičenja CSP rešavača. Sistemi koji koriste svođenje na SAT ili neku od bliskih metoda su opisani u poglavlju 2.3.8.

2.2.4.1 Sistemi zasnovani na pretrazi i propagiranju

Veliki broj sistema je zasnovan na korišćenju kombinacije neke vrste pretrage i algoritama za propagiranje ograničenja, pri čemu ti sistemi imaju posebno razvijene algoritme za obradu globalnih ograničenja. Najčešće sistemi koriste neku verziju pretrage sa povratkom i algoritme za postizanje lučne konzistencije.

Gecode [110] je skup alatki koji olakšava razvoj CP sistema. U okviru distribucije postoji i istoimeni rešavač, koji je osvojio veliki broj medalja na takmičenjima CSP rešavača.

Mistral [60] je CP biblioteka koja na raspolaganju nudi različite algoritme pretrage i propagiranja ograničenja. Postoji istoimeni rešavač koji koristi ovu biblioteku i koji se pokazao vrlo uspešnim na takmičenjima CSP rešavača.

Još neki poznati sistemi/biblioteke ovog tipa su *Abscon* [93] i `choco` [70].

2.2.4.2 Sistemi zasnovani na logičkom programiranju uz ograničenja

Sistemi u ovoj grupi koriste jezike koji predstavljaju ekstenzije PROLOG-a i implementiraju rešavače za te jezike. Najpoznatiji sistemi ovog tipa su `BProlog` [136] i `ECLiPSe` [5], kao i rešavači implementirani u okviru distribucija nad jezicima `SWI-Prolog` [133] i `CHIP` [13].

2.2.4.3 Sistemi koji koriste lokalnu pretragu

Iako se lokalna pretraga u ogromnoj meri koristi za rešavanje pojedinačnih problema, njena implementacija u okviru sistema za rešavanje problema u nekom opštem obliku nije rasprostranjena.

Sistem `fzn_izplus` [123] je više puta učestovao i osvajao medalje na takmičenjima CSP rešavača. Ovaj sistem pri rešavanju problema COP prvo nalazi rešenje korišćenjem pretrage i propagiranja, a potom pokušava da unapredi rešenje korišćenjem lokalne pretrage.

Sistem `fzn-oscar-cbls` [22] omogućava korišćenje lokalne pretrage za rešavanje problema u MiniZinc jeziku. Moguće je koristiti pohlepnu ili tabu pretragu za traženje rešenja.

2.2.4.4 Ostali sistemi

Postoji veliki broj sistema koji koriste tehnike iz drugih oblasti za rešavanje problema CSP (na primer, metode linearnog programiranja, celobrojnog programiranja, linearne algebre).

IBM ILOG CPLEX Optimization Studio[6] je sistem koji omogućava specifikaciju modela optimizacionih problema i rešavanje tih specifikacija pomoću metoda matematičkog programiranja i programiranja ograničenja. Ovaj sistem je u širokoj upotrebi u različitim oblastima.

`OR-Tools` [105] je sistem alatki iz oblasti operacionih istraživanja koji omogućuje rešavanje problema CSP.

Rešavač `mzn-g12mip` u okviru MiniZinc distribucije takođe spada u ovu grupu sistema, ali više detalja o njemu nije dostupno.

2.2.5 Rešavanje problema COP metodom grananja i ograničavanja

Metoda *grananje i ograničavanje* (eng. *branch and bound*) [4, 11] se koristi za rešavanje problema COP (definicija ovog problema data je u poglavlju 2.1.5). Prilikom rešavanja ovog problema, pored zadovoljenja svih ograničenja potrebno je naći i maksimalnu/minimalnu vrednost funkcije cilja $f : D_1 \times \ldots \times D_n \to \mathbb{R}$. U ovom poglavlju, bez smanjenja opštosti ćemo pretpostaviti da je potrebno naći maksimalnu vrednost (pogledati poglavlje 2.1.5 za objašnjenje).

Opšti oblik metode grananja i ograničavanja. Prvi korak ove metode je određivanje donje granice vrednosti funkcije cilja. To se postiže ili nalaženjem nekog rešenja problema i izračunavanjem odgovarajuće vrednosti funkcije cilja za to rešenje, ili odabirom neke izuzetno male vrednosti za donju granicu funkcije cilja (na primer, $-\infty$). Potom se vrši *grananje*, odnosno deljenje (dekompozicija) početnog problema na podprobleme i određivanje redosleda rešavanja ovih podproblema. Podproblemi imaju isti skup ograničenja kao početni problem, a domen svakog od podproblema

[6] `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`

je podskup domena početnog problema. Unija domena svih podskupova jednaka je domenu početnog problema. Za svaki podproblem rešava se *relaksacija* tog podproblema, tj. problem koji ima skup rešenja koji je nadskup skupa rešenja nerelaksiranog problema, ali koji je obično značajno lakši za rešavanje (može se na primer izbaciti neko nepovoljno ograničenje ili celobrojni domeni zameniti realnim). Postoje tri moguća ishoda rešavanja relaksacije problema: 1) ako se nađe rešenje relaksacije bolje od najboljeg dosadašnjeg rešenja koje je istovremeno i rešenje nerelaksiranog problema, onda se to rešenje proglašava najboljim do sada nađenim i prelazi se na rešavanje narednog podproblema; 2) ako se ne pronađe rešenje ili ako se pronađe rešenje koje nije bolje od do sada najboljeg, prelazi se na rešavanje narednog podproblema; 3) ako se pronađe rešenje relaksacije koje nije rešenje nerelaksiranog problema, onda se vrši ponovno grananje.

Grananje i ograničavanje kao modifikacija pretrage sa povratkom. U oblasti programiranja ograničenja, grananje i ograničavanje se najčešće definiše na specijalan način – kao modifikacija pretrage sa povratkom. To je samo jedna od mnogo mogućih varijanti algoritma grananja i ograničavanja, i ovde opisujemo tu varijantu.

Pre primene grananja i ograničavanja, potrebno je odrediti *heurističku funkciju* $h : \mathcal{P}(D_1) \times \ldots \times \mathcal{P}(D_n) \to \mathbb{R} \cup \{\infty\}$ ($\mathcal{P}(X)$ je partitivni skup skupa X), koja omogućava da se *efikasno* ograniči vrednost funkcije f odozgo. Ideja je da se vrednost funkcije h može odrediti i u situaciji kada samo deo promenljivih ima dodeljene vrednosti (*parcijalno rešenje*), i to je razlog zašto je ona definisana nad podskupovima domena. Vrednost funkcije je *gornja granica* vrednosti funkcije f na svim rešenjima koja se mogu dobiti proširivanjem parcijalnog rešenja do punog rešenja. Funkcija h omogućava da se prilikom izgradnje rešenja napuste parcijalna rešenja koja se ne mogu proširiti do kompletnih rešenja sa vrednošću funkcije f većom od do tog trenutka najbolje pronađene vrednosti. U opštem slučaju, određivanje funkcije h nije lak zadatak i zahteva dobro poznavanje problema koji se rešava. Prvo, poželjno je izabrati funkciju tako da se njene vrednosti mogu brzo izračunati, jer se izračunavanje vrednosti funkcije h vrši često. Drugo, što su manje razlike između vrednosti ove funkcije i odgovarajućih vrednosti funkcije cilja, veći je i broj parcijalnih rešenja koja se napuštaju. Prethodna dva zahteva su vrlo često u suprotnosti, pa je potrebno napraviti balans među njima pri odabiru heurističke funkcije.

Dva osnovna svojsta koja funkcija h mora da zadovolji su:

- Monotonost: ako je za svako $i \in \{1, \ldots, n\}$ ispunjeno $E_i \subseteq F_i \subseteq D_i$, onda je $h(E_1, \ldots, E_n) \leq h(F_1, \ldots, F_n)$.
- Ograničavanje: $f(d_1, \ldots, d_n) \leq h(\{d_1\}, \ldots, \{d_n\})$.

Prvo svojstvo osigurava da se smanjivanjem domena promenljivih samo može smanjiti ili zadržati ista gornja granice funkcije f. U slučaju kada se ustanovi da se parcijalno rešenje ne može proširiti do rešenja koje će imati bolju vrednost funkcije cilja od već pronađenog rešenja, ovo svojstvo garantuje da takvo (prošireno) rešenje stvarno neće postojati, tj. da se neće dobiti veća vrednost funkcije h. Drugo svojstvo osigurava da funkcija h uvek uzima vrednosti veće ili jednake od vrednosti funkcije f. Ponekad se od funkcije h zahteva da zadovoljava dodatno svojstvo da u optimumu uzima istu vrednost kao i funkcija f [111].

Primer 11 (*Problem ranca.*)**.** *Problem koji se razmatra je već opisan u primeru 5. Funkcija cilja problema je* $f(x_1, \ldots, x_n) = \sum_{i=1}^{n} v_i x_i$ *i potrebno je naći rešenje* $(x_1, \ldots, x_n)$ *sa maksimalnom vrednošću ove funkcije, koje zadovoljava uslov* $\sum_{i=1}^{n} z_i x_i \leq Z$.

Razmotrimo privremeno varijantu problema ranca gde se predmeti mogu seći tako da se u ranac može dodati proizvoljan deo bilo kog predmeta, tj. sve promenljive $x_1, \ldots, x_n$ *imaju domen* $[0, 1]$. *Ta varijanta se lako rešava: predmeti se prvo sortiraju u opadajućem poretku po količniku vrednosti i zapremine. Redom se uzimaju celi predmeti iz sortiranog poretka tako da zbir njihovih zapremina ne prevaziđe zapreminu ranca. Dakle, uzima se najvredniji predmet po jedinici zapremine, pa potom sledeći najvredniji po jedinici zapremine, itd. Ukoliko posle poslednjeg uzetog predmeta ima još prostora u rancu a nisu svi predmeti iskorišćeni, onda se od narednog predmeta uzima samo deo koji može da popuni preostalu zapreminu ranca. Upravo opisan postupak rešavanja ove varijante problema ranca će nam poslužiti u formiranju heuristike originalnog problema.*

Vratimo se na razmatranje originalnog problema. Kao heurističku funkciju se uzima funkcija cilja originalnog problema, ali uz dve modifikacije. Prvo, promenljive sada imaju realne domene, tj. sve $x_1, \ldots, x_n$ *imaju domen* $[0, 1]$. *Drugo, neka promenljive* $x_1, \ldots, x_k$ *već imaju redom dodeljene vrednosti* $a_1, \ldots, a_k$ *iz skupa* $\{0, 1\}$. *Uvedimo prvo pomoćnu funkciju* $f'(x_{k+1}, \ldots, x_n) = max\,\{f(0, \ldots, 0, x_{k+1}, \ldots, x_n)\}$, *gde* $x_{k+1}, \ldots, x_n$ *uzimaju vrednosti iz segmenta* $[0, 1]$ *i pri tome mora biti ispunjeno* $\sum_{i=k+1}^{n} z_i x_i \leq Z - \sum_{i=1}^{k} z_i a_i$ *(suma zapremina predmeta koji se uzimaju ne sme biti veća od preostale zapremine ranca). Napomenimo da smo ovde fiksirali redosled pro-*

menljivih, tj. funkciju f' smo definisali samo nad poslednjih $n-k$ promenljivih, gde je k vrednost između 0 i $n-1$. U opštem slučaju redosled se ne mora fiksirati, ali je to ovde urađeno da bi se dobio jednostavniji izraz za funkciju f'. Dakle, funkcija f' izračunava kolika je maksimalna vrednost koja se može dobiti dodeljivanjem vrednosti iz segmenta $[0,1]$ promenljivama koje nemaju dodeljene vrednosti, a da pri tome suma zapremina uzetih predmeta ne bude veća od preostale zapremine $Z-\sum_{i=1}^{k} z_i a_i$. Maksimum se dobija na način opisan u prethodnom pasusu – redom se uzimaju predmeti sa najboljim količnikom vrednosti i kapaciteta, do trenutka kada se popuni ranac. Heuristika se definiše kao zbir $h(a_1,\ldots,a_k,x_{k+1},\ldots,x_n)=\sum_{i=1}^{k} a_i v_i + f'(x_{k+1},\ldots,x_n)$. Dakle, uzimajući da su promenljive $x_1,\ldots,x_k$ već dobile vrednosti, funkcija h izračunava kolika je maksimalna vrednost koja se može dobiti dodeljivanjem vrednosti iz segmenta $[0,1]$ preostalim promenljivama, a da se sa svim uzetim predmetima ne prevaziđe zapremina ranca.

Lako je uvideti da predložena funkcija h zadovoljava uslove monotonosti i ograničavanja. Prvo, maksimalna vrednost koja se može dobiti popunjavanjem ranca u slučaju kada se od svakog predmeta može uzeti i deo, veća je ili jednaka od maksimalne vrednosti koja se može dobiti uzimanjem samo celih predmeta. Drugo, funkcija h na vrednostima promenljivh iz skupa $\{0,1\}$ uzima iste vrednosti kao i funkcija f, pa je nejednakost iz uslova ograničavanja trivijalno ispunjena.

Pomenuto razmatranje realnih umesto celobrojnih domena u celobrojnom programiranju predstavlja relaksaciju problema. Pored korišćenja ove tehnike u kombinaciji sa metodom grananja i ograničavanja, problem ranca se vrlo često rešava i pomoću tehnika dinamičkog programiranja*. Tehnike ove oblasti su dosta proučavane i upotrebljavane u rešavanju raznoraznih problema [111, 108]. U slučaju rešavanja problema ranca metodama dinamičkog programiranja, formira se tabela u kojoj se čuvaju maksimalne dobijene vrednosti za sve manje zapremine od one koju treba popuniti i onda se te vrednosti iskorišćavaju u razmatranju popunjavanja ranca zapremine Z.*

Pseudokod algoritma grananja i ograničavanja dat je na slici 2.7 i on predstavlja modifikaciju naivne pretrage sa povratkom prilagođenu optimizacionim problemima – ovaj algoritam se može učiniti efikasnijim istim tehnikama kao i u slučaju rešavanja problema CSP. Sa ciljem pojednostavljivanja koda, promenljive `granica` i `rešenje` su deklarisane kao globalne – one redom čuvaju najveću pronađenu vrednost funkcije f i rešenje za koje je postignuta ta vrednost. Ako je pronađeno rešenje (`Buduće` = $\emptyset$) onda se proverava da li ono daje veću vrednost funkcije cilja od najveće do tog

trenutka – ukoliko to jeste slučaj, pamte se informacije o novom najboljem rešenju. Funkcija h određuje da li neko parcijalno rešenje potencijalno može da se proširi tako da se dobije veća vrednost funkcije cilja od najveće do sada nađene (`granica`). Novi rekurzivni poziv ne garantuje nalaženje boljeg rešenja problema, jer funkcija h u opštem slučaju samo procenjuje, a ne izračunava precizno vrednost gornje granice funkcije f.

```
granica = −∞, rešenje = ∅ // globalne promenljive
function BB (Buduće, Prošle)
  if Buduće = ∅ && f (vrednosti (Prošle)) > granica
    granica = f (vrednosti (Prošle));
    rešenje = Prošle;
    return;
  izaberi neku promenljivu x iz skupa Buduće
  for each vrednost v iz domena D(x)
    if Proveri_konzistenciju ({x = v} ∪ Prošle)
      if h (vrednosti (Prošle)) > granica)
        BB (Buduće \ {x}, Prošle ∪ {x = v});
```

Slika 2.7: Algoritam grananja i ograničavanja. Skup domena promenljivih i ograničenja (`C`) se smatraju globalnim promenljivama. Funkcija `vrednosti` vraća n-torku koja se sastoji od trenutno dodeljenih vrednosti promenljivama i celih domena promenljivih u slučaju da im nisu dodeljene vrednosti.

Teorema 2 (*Korektnost grananja i ograničavanja.*)**.** *Grananje sa ograničavanjem je algoritam koji se uvek završava. Ako početni skup ograničenja ima rešenje, onda se po završetku algoritma u promenljivoj* `rešenje` *nalazi rešenje sa maksimalnom vrednošću funkcije f. Ako rešenje ne postoji, onda promenljiva* `rešenje` *ima vrednost $\emptyset$.*

2.3 Pregled metoda rešavanja problema CSP svođenjem na SAT

U ovom poglavlju ćemo opisati problem SAT a potom i postupak svođenja problema CSP na problem SAT. Zatim će biti opisana kodiranja koja se koriste pri svođenju na SAT i to: direktno kodiranje, kodiranje podrške, kodiranje uređenja i logaritamsko kodiranje. Na kraju će biti navedeni postojeći sistemi za kodiranje problema CSP na problem SAT.

2.3.1 Problem SAT i neke tehnike njegovog rešavanja

Problem iskazne zadovoljivosti, SAT problem (eng. *Boolean Satisfiability Problem*) [37], je specijalan slučaj problema CSP. Data je formula F u konjunktivnoj normalnoj formi:

$$F = \bigwedge_{i=1}^{m} c_i \quad \text{gde je} \quad c_i = \bigvee_{j=1}^{k(i)} l_{i,j}.$$

Elementi disjunkcija su ili iskazne promenljive (na primer, l) ili negacije iskaznih promenljivih ($\neg l$) i zovu se *literali* (eng. *literals*). *Suprotan literal* (eng. *opposite literal*) literala l označavamo sa $\neg l$. Ako je literal promenljiva, njemu suprotan literal je negacija te promenljive, a ako je literal negacija promenljive, njemu suprotan literal je sama ta promenljiva. Disjunkcija c_i literala se zove *klauza* (eng. *clause*). Zadatak je odrediti da li je formula F *zadovoljiva* (eng. *satisfiable*) – utvrditi da li postoji dodela iskaznih vrednosti 1 i 0 promenljivama tako da je u svakoj klauzi barem jedan literal tačan (l je tačan ako je l dodeljena vrednost 1, a $\neg l$ je tačan ako je l dodeljena vrednost 0). Postupak utvrđivanja zadovoljivosti formule zove se i *rešavanje* formule. Za vrednosti iskaznih promenljivih mogu se koristiti i oznake *tačno/netačno* (eng. *true/false*) ili $\top/\bot$, ali su u ovom tekstu korišćene vrednosti $1/0$. Problem SAT je specijalan slučaj problema CSP u kome sve promenljive imaju isti domen $\{0, 1\}$, a jedina ograničenja koja se zadaju su klauze nad tim promenljivama.

Niz literala se zove *valuacija*. Za valuaciju se kaže da je *neprotivrečna* (eng. *consistent*) ako ne sadrži dva suprotna literala. Neprotivrečna valuacija jednoznačno preslikava skup promenljivih u skup istinitosnih vrednosti $\{0, 1\}$, tj. promenljivama dodeljuje vrednosti (ako je u valuaciju uključena promenljiva onda se njoj dodeljuje vrednost 1, ako je uključena njena negacija, promenljivoj se dodeljuje vrednost 0). Cilj rešavanja SAT formule može se formulisati i kao određivanje neprotivrečne valuacije koja za svaku promenljivu uključuje ili nju, ili njenu negaciju. Ako postoji, takva valuacija je rešenje SAT formule.

Literal l je *tačan u valuaciji* M (označavamo sa $M \models l$) ako je njen član. Literal l je *netačan u valuaciji* M (označavamo sa $M \models \neg l$) ako je $\neg l$ njen član. Klauza c je *tačna u valuaciji* M (označavamo sa $M \models c$) ako je bar jedan literal te klauze tačan u valuaciji M. Klauza c je *netačna u valuaciji* M (označavamo sa $M \models \neg c$) ako su svi literali te klauze netačni u valuaciji M. Formula F je *tačna u valuaciji* M (označavamo sa $M \models F$) ako su sve klauze te formule tačne u valuaciji M. Formula F je *netačna u valuaciji* M ako je barem jedna klauza te formule netačna

u valuaciji M. Primetimo da pojam valuacije obuhvata i parcijalne valuacije, tj. ne mora svaka promenljiva da ima dodeljenu vrednost u nekoj valuaciji. Valuacija je *model*[7] formule (klauze, literala) ako je neprotivrečna i ako je formula (klauza, literal) u njoj tačna. Primetimo da i model može biti parcijalna valuacija – on se može proširiti do potpune valuacije dodelom na proizvoljan način vrednosti 0 ili 1 svakoj od promenljivih koja nema dodeljenu vrednost modelom.

Formula F *povlači* (eng. *entails*) formulu F' (klauzu c, literal l) ukoliko je svaki model formule F istovremeno i model formule F' (klauze c, literala l). Za prethodno koristimo oznaku $F \models F'$ ($F \models c$, $F \models l$) i kažemo da je formula F' (klauza c, literal l) logička posledica formule F. Ako je za dve formule ispunjeno da je svaki model jedne formule istovremeno i model druge formule, onda kažemo da su te dve formule *logički ekvivalentne*. Dve formule su *ekvizadovoljive* (eng. *equisatisfiable*) ako je ispunjeno: prva je zadovoljiva akko je druga zadovoljiva.

Programi koji na ulazu primaju i potom rešavaju iskaznu formulu se zovu *SAT rešavači* (eng. *SAT solvers*). Opšteprihvaćen format za SAT formulu u KNF-u je DIMACS[8]. U poslednjih par decenija razvijen je veliki broj SAT rešavača a jednom godišnje se organizuje i konferencija "International Conference on Theory and Applications of Satisfiability Testing"[9] posvećena svim temama vezanim za SAT rešavanje. U okviru te konferencije organizuju se i takmičenja koja za cilj imaju razvoj što je moguće efikasnijih SAT rešavača kao i programa za rešavanje problema bliskih problemu SAT.

2.3.1.1 DPLL procedura - opis u vidu pseudokoda

Zbog povezanosti problema SAT i CSP, mnoge metode prvobitno razvijene za SAT rešavače se koriste i za CSP rešavače, i obrnuto. Većina savremenih potpunih SAT rešavača koristi *Dejvis-Putnam-Logeman-Loveland-ovu proceduru*, skraćeno DPLL, koja je verzija pretrage sa povratkom i na kojoj je zasnovana već opisana pretraga kod CSP rešavača. Pseudokod ove procedure je prikazan na slici 2.8 i predstavlja modifikaciju algoritma koji je opisao Filip Marić ([90]). Sa F je označena iskazna formula (skup klauza) čiju zadovoljivost je potrebno utvrditi, dok $F[l \to 1]$ označava formulu dobijenu zamenom l sa 1 i $\neg l$ sa 0 (analogno za $F[l \to 0]$), i uprošćavanjem posle toga, tj. uklanjanje zadovoljenih klauza i netačnih literala. Literal

[7] Ovaj pojam treba razlikovati od pojma uvedenog kod CSP rešavanja, gde termin model označava predstavljanje problema CSP skupom promenljivih, njihovih domena i ograničenja.

[8] `ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/doc/satformat.dvi`

[9] `http://www.satisfiability.org/`

je *jedinstvenog polariteta* ako se pojavljuje u formuli a njegova negacija ne – u situaciji kada postoji takav literal dodeljuje se vrednost odgovarajućoj promenljivoj tako da literal bude tačan i time zadovoljavaju sve klauze koje sadrže taj literal. Ako klauza sadrži samo jedan literal onda joj se dodeljuje vrednost tako da literal bude tačan, a time i klauza postaje tačna.

function DPLL (F)
  if F prazna
    return SAT
  else if F sadrži praznu klauzu
    return UNSAT
  else if F sadrži literal l jedinstvenog polariteta
    return DPLL ($F[l \to 1]$)
  else if F sadrži klauzu sa jednim literalom $[l]$
    return DPLL ($F[l \to 1]$)
  else
    izaberi literal l koji se javlja u F
    if DPLL ($F[l \to 1]$) == SAT
      return SAT
    else
      return DPLL ($F[l \to 0]$)

Slika 2.8: DPLL algoritam.

Dodela vrednosti promenljivoj (1 ili 0) je dodavanje odgovarajućeg literala te promenljive (l ili $\neg l$) u valuaciju, dok uklanjanjem literala iz valuacije promenljivoj prestaje da bude dodeljena vrednost. DPLL procedura kreće od prazne valuacije i pokušava da je proširi dodavanjem novih literala. Svako dodavanje je predstavljeno novim rekurzivnim pozivom.

Teorema 3 (*Korektnost DPLL procedure.*)**.** *DPLL procedura se uvek završava. Ako početna formula* `F` *ima rešenje, onda DPLL procedura vraća odgovor SAT. U suprotnom, DPLL procedura vraća odgovor UNSAT.*

Opisana rekurzivna verzija algoritma je neefikasna i neupotrebljiva za veće formule, jer zahteva kreiranje nove uprošćene formule i njeno prosleđivanje svakim rekurzivnim pozivom (ovo je vrlo zahtevno jer formule mogu biti jako velike).

2.3.1.2 DPLL procedura - opis u vidu sistema promene stanja

Drugi način opisa DPLL procedure je pomoću pravila promena stanja. Prednost ovakvog opisa je to što su SAT rešavači predstavljeni kao matematički objekti o kojima se može jednostavno formalno rezonovati. Takođe, ovakav opis potpuno i precizno opisuje najviši nivo arhitekture SAT rešavača. Nedostaci ovakvog opisa su njegova apstraktnost i udaljenost od konkretne implementacije. Tekst se zasniva na opisu Filipa Marića [90], a za zapis pravila je korišćen format zasnovan na radu Krstića i Goela [79]. Preciznije, pravila su oblika:

`Ime pravila:`

$$\frac{uslov_1, \quad \ldots, \quad uslov_k}{efekat}$$

Uslovi koji moraju biti ispunjeni su navedeni iznad linije, a efekat pravila na stanje procedure je naveden ispod linije.

Za razliku od procedure prikazane na slici 2.8 gde se u rekurzivnim pozivima menja formula koja se rešava, savremeni SAT rešavači eksplicitno čuvaju tekuću parcijalnu valuaciju a ne menjaju formulu koju rešavju. U valuaciji se razlikuju dve vrste literala: *pretpostavljeni literali* (eng. *decision literals*) i *izvedeni literali* (eng. *implied literals*). Prve ćemo označavati dodavanjem uspravne crte ispred njih (na primer $|\, l$). Stanje rešavača je određeno tekućom parcijalnom valuacijom M i formulom F čiju je zadovoljivost potrebno utvrditi. Valuacije ćemo predstavljati listama literala. Skup L označava skup literala zatvoren za negaciju, a definisan je nad skupom promenljivih koje se javljaju u formuli F.

Sledeća dva pravila su dovoljna da se opiše procedura koja koristi samo pretragu.

`Decide`:

$$\frac{l \in L, \quad l, \neg l \notin M}{M := M \mid l}$$

`Backtrack`[10]:

[10] Uslov da se u delu valuacije M'' ne nalazi nijedan pretpostavljen literal označen je sa $(decisions\ M'') = []$.

$$\frac{M = M' \mid l\ M'', \quad (decisions\ M'') = [], \quad M \models \neg F}{M := M'\ \neg l}$$

Pravilo `Decide` dodaje u valuaciju literal l koji ispunjava uslov da prethodno ni on a ni njegova negacija nisu dodati u valuaciju. Literali koji se razmatraju su najčešće oni koji se javljaju u polaznoj formuli. Literal se dodaje kao pretpostavljeni literal i kaže se da je izvršeno *grananje* (eng. *branching*). Postoji veliki broj heuristika za odabir literala prilikom grananja. Grananje odgovara biranju promenljive i vrednosti koja će joj biti dodeljena u slučaju CSP rešavača. U razvoju SAT rešavača je posvećena značajno veća pažnja konstrukciji dobrih algoritama grananja nego što je to slučaj kod CSP rešavača.

Pravilo `Backtrack` se primenjuje u trenutku kada formula F postane netačna u valuaciji M. Kažemo da je u toj situaciji došlo do *konflikta*[11], a klauze formule F koje su netačne se zovu *konfliktne klauze* (eng. *conflict clauses*). Poslednji pretpostavljeni literal l i svi literali iza njega se uklanjaju iz valuacije i zatim se literal $\neg l$ dodaje u valuaciju kao izvedeni literal.

Završno stanje se razlikuje od ostalih po tome što se u njemu ne može primeniti nijedno pravilo. Ako je u završnom stanju ispunjeno $M \models \neg F$ onda je formula nezadovoljiva. U suprotnom, formula je zadovoljiva i M je jedan model te formule.

Redosled pravila nije definisan, ali se u praksi bolja efikasnost postiže ako se pravilu `Backtrack` da veći prioritet, tj. kada se ono primenjuje kad god je to moguće.

2.3.1.3 Poboljšanja DPLL procedure

Spomenućemo nekoliko tehnika koje se koriste za poboljšavanje efikasnosti DPLL procedure. To su tehnike koje su ključne za efikasnost i zato su implementirane u većini modernih SAT rešavača. Postoje mnogo detaljniji prikazi ovih tehnika ([19, 90]), a ovde komentarišemo sličnosti i razlike sa već opisanim CSP tehnikama.

Jedinične klauze. Ako je za neku klauzu i valuaciju M ispunjeno da je tačno jedan literal l te klauze nedefinisan u M dok su svi ostali literali netačni u toj valuaciji, onda tu klauzu zovemo *jedinična klauza* (eng. *unit clause*) a literal l *jedinični literal*. Da bi jedinična klauza bila tačna, potrebno je valuaciju proširiti jediničnim literalom te klauze. Mehanizam koji služi za obradu jediničnih klauza se zove

[11] Ovaj pojam ima drugačije značenje od pojmova uvedenih kod pretrage sa povratkom kod CSP rešavača i kod lokalne pretrage.

propagiranje jediničnih klauza (eng. *unit propagation*) i ova tehnika se iterativno primenjuje dok svi jedinični literali ne dobiju svoje vrednosti.

Propagiranje jediničnih klauza se opisuje sledećim pravilom.

`Unit Propagate`:

$$\frac{l \vee l_1 \vee \ldots \vee l_k \in F, \quad \neg l_1, \ldots, \neg l_k \in M, \quad l, \neg l \notin M}{M := M\ l}$$

Sa stanovišta efikasnosti, neprihvatljivo je prolaziti kroz sve klauze da bi se utvrdilo da li među njima ima jediničnih. Shema koja je danas u upotrebi se zove *shema dva posmatrana literala* (eng. *two-watch literal scheme*). U ovom pristupu se svakoj klauzi pridružuju dva literala te klauze koja se "posmatraju" a provera da li je klauza postala jedinična se vrši samo kada jedan od dva posmatrana literala postane netačan. Najčešća konvencija je da se za posmatrane literale uzmu prvi i drugi literal u klauzi.

Literali jedinstvenog polariteta. Već je rečeno šta su literali jedinstvenog polariteta. Ovi literali se dodaju u valuaciju kao izvedeni literali. Određivanje literala jedinstvenog polariteta je skupa operacija, pa se kod većine savremenih SAT rešavača dodavanje ovih literala primenjuje samo u fazi preprocesiranja, (tj. pre nego što se krene sa pretragom) dok se kod pojedinih rešavača u potpunosti izbegava ova tehnika. Sledeće pravilo se uvodi u opisu rešavača koji koriste ovaj mehanizam.

`Pure literal`:

$$\frac{l \in F, \quad \neg l \notin F, \quad l, \neg l \notin M}{M := M\ l}$$

Učenje. Mehanizam *učenja klauza* (eng. *clause learning*) omogućava dodavanje novih klauza skupu klauza formule, i ove klauze se zovu *naučene klauze* (eng. *learned clauses*). Naučene klauze su posledica početnog skupa klauza i za cilj imaju izbegavanje konflikata do kojih se već dolazilo. Ova tehnika odgovara pretrazi sa obeležavanjem kod CSP rešavača. Sledeće pravilo se uvodi u opisu rešavača koji koriste ovaj mehanizam.

`Learn`:

$$\frac{F \models c, \quad c \subseteq L}{F := F \ \cup \ c}$$

Uslov $F \models c$ zahteva da je klauza c logička posledica formule F i obezbeđuje da je nova formula ($F := F \ \cup \ c$) logički ekvivalentna sa polaznom. Uslov $c \subseteq L$ zahteva da svi literali iz c pripadaju skupu literala L i time se obezbeđuje da se ne uvode novi literali (ovo je bitan uslov da bi se obezbedilo zaustavljanje procedure). Ovde ne razmatramo način na koji je moguće odrediti klauzu c ([90]).

Zaboravljanje. Tokom rešavanja se učenjem skup klauza povećava i u nekom trenutku može postati prevelik za efikasan rad rešavača. Tada je potrebno ukloniti neke klauze ali uz uslov da nova formula mora biti ekvizadovoljiva sa polaznom formulom. Zato se obično brišu samo klauze iz skupa naučenih klauza. Zaboravljanje klauza se opisuje sledećim pravilom.

`Forget`:

$$\frac{F \setminus c \models c}{F := F \setminus c}$$

Pretraga sa skokom unazad vođena analizom konflikta. *Pretraga sa skokom unazad vođena analizom konflikta* (eng. *conflict-driven backjumping*) predstavlja modifikaciju DPLL procedure. Ona se zasniva na korišćenju već pomenutih konfliktnih klauza, a odgovara prelasku sa pretrage sa povratkom u pretragu sa skokom unazad u slučaju CSP rešavača. Ovom tehnikom se omogućava brisanje više pretpostavljenih literala iz valuacije odjednom, sve do poslednjeg pretpostavljenog literala koji je zaista učestvovao u konfliktu. Time se izbegava trošenje vremena na proširivanje valuacije koja sigurno ne dovodi do rešenja.

Pre skoka potrebno je odrediti *klauzu povratnog skoka* (eng. *backjump clause*). Ova klauza treba da odgovara promenljivama koje su dovele do konflikta. Po njenoj konstrukciji, literali valuacije se uklanjaju sve dok klauza povratnog skoka ne postane jedinična klauza u odnosu na tu valuaciju. Njen jedinični literal se postavlja na vrh tekuće valuacije kao izvedeni literal i proces pretrage se nastavlja. Povratni skokovi se opisuju sledećim pravilom.

`Backjump`:

$$\frac{c = l \vee l_1 \vee \ldots \vee l_k, \quad F \models c, \quad M = M' \mid d\ M'', \quad \neg l_1, \ldots, \neg l_k \in M', \quad \neg l \in M''}{M := M'\ l}$$

Analiza konflikta. Sa ciljem da klauze povratnog skoka zaista opišu konflikt do koga je došlo i time doprinesu najvećem mogućem odsecanju prostora pretrage, sprovodi se postupak koji se zove *analiza konflikta* (eng. *conflict analysis*). Postoji više različitih strategija za analizu konflikta a njima se u istoj situaciji može dobiti različita klauza povratnog skoka. Najveći broj strategija se zasniva na sledećoj tehnici. Prvo se odredi konfliktna klauza c, tj. klauza koja je postala netačna u tekućoj valuaciji. Drugo, za svaki izvedeni literal iz c se pronalazi klauza koja je bila uzrok propagiranja literala l. Ove klauze se zovu *razlozi propagiranja* (eng. *reason clauses*). Svaki od izvedenih literala u klauzi c se zamenjuje ostalim literalima iz klauze koja predstavlja uzrok propagiranja tog literala. Prvom i drugom delu opisane strategije odgovaraju redom sledeća dva pravila.

`Conflict`:

$$\frac{l_1 \vee \ldots \vee l_k \in F, \quad \neg l_1, \ldots, \neg l_k \in M}{c := l_1 \vee \ldots \vee l_k}$$

`Explain`:

$$\frac{\neg l \in c, \quad l \vee l_1 \vee \ldots \vee l_k \in F, \quad \neg l_1, \ldots, \neg l_k \prec^M l}{c := (c \setminus \neg l) \vee l_1 \vee \ldots \vee l_k}$$

Uslov $\neg l_1, \ldots, \neg l_k \prec^M l$ označava da literali $\neg l_1, \ldots, \neg l_k$ prethode literalu l u valuaciji M. Posle prvog pravila, drugo se primenjuje sve dok ne bude ispunjen neki od unapred određenih uslova prekida i tada se trenutna klauza c analize koflikta proglašava klauzom povratnog skoka. Najčešće korišćena strategija kod savremenih SAT rešavača za određivanje uslova prekida se zove *prva tačka jedinstvene implikacije* (eng. *first unique implication point*), skraćeno firstUIP. Kod ove strategije se proces analize konflikta prekida kada se dogodi da iza poslednjeg pretpostavljenog literala u M postoji tačno jedan literal klauze c. Više detalja o mogućim strategijama je dostupno u mnogobrojnoj literaturi (na primer, [90, 79]). Spomenimo da je poželjno i da klauza povratnog skoka ima što je manji broj literala, jer takva klauza dovodi do većih odsecanja prostora pretrage. Procedure za minimizaciju klauze povratnog skoka detaljno su opisane u pomenutoj literaturi.

Otpočinjanje iznova. *Otpočinjanje iznova* (eng. *restarting*) omogućava da se s vremena na vreme poništi skoro celokupna valuacija (preciznije, deo valuacije počev od prvog pretpostavljenog literala) i proces pretrage krene od početka. Ova tehnika može da dovede do bržeg rešavanja formule u slučaju kada se krene u dopunjavanje valuacije nekom novom, bržom putanjom. Otpočinjanje iznova ne treba neograničeno koristiti, jer bi u tom slučaju moglo da se ugrozi zaustavljanje procedure. Ova tehnika se koristi i kod CSP rešavača, ali u značajno manjoj meri nego kod SAT rešavača. Otpočinjanje iznova se opisuje sledećim pravilom.

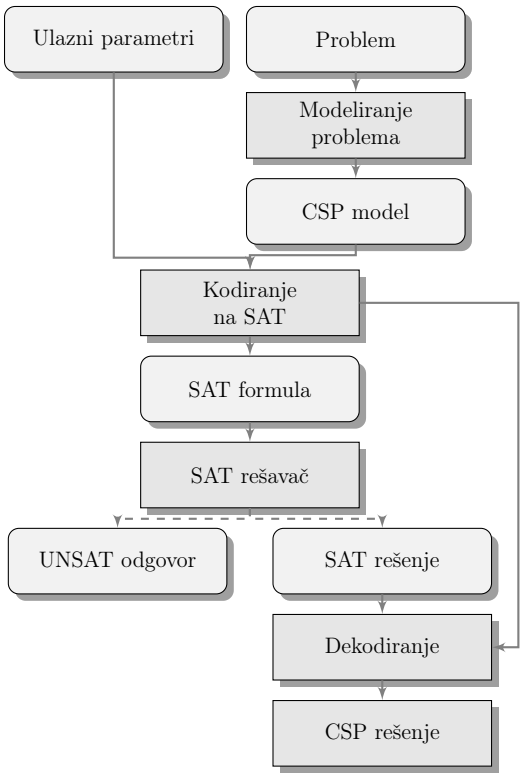
$$\texttt{Restart}: \quad \frac{\top}{M := M^{[0]}}$$

U pravilu $M^{[0]}$ označava prefiks valuacije M pre pojave prvog pretpostavljenog literala.

2.3.2 Postupak svođenja na SAT

Postupak rešavanja konačnog linearnog problema CSP svođenjem na SAT sastoji se iz više faza i prikazan je na slici 2.9. Prvo je potrebno modelirati problem, odnosno formulisati problem koji se rešava kao problem CSP, određivanjem promenljivih, njihovih domena i ograničenja. Obično postoji više različitih načina da se to uradi, a od odabranog CSP modela zavisi koje metode i rešavači mogu da se upotrebe u rešavanju. Potom se CSP model zajedno sa ulaznim parametrima problema prevodi u problem SAT, tj. u promenljive i klauze SAT problema (i za ovo postoji više različitih načina). Neki od mnogobrojnih SAT rešavača se pokreće da reši SAT formulu. Ako SAT rešavač utvrdi da je problem nezadovoljiv, onda je i početni problem nezadovoljiv. Ako se utvrdi da je problem zadovoljiv, onda se rešenje SAT problema prevodi inverznom transformacijom u rešenje početnog problema. Značajan razlog uspeha rešavanja problema CSP svođenjem na SAT leži u velikom broju slobodnih, besplatnih i javno dostupnih SAT rešavača koji mogu da reše SAT formule koje se sastoje od miliona klauza. Pošto su slobodni, ovi rešavači se mogu prilagoditi različitim potrebama.

SAT rešavači se mogu efikasno upotrebiti za rešavanje raznih problema CSP ali je preduslov da formula koju dobijaju ne bude prevelika (iako veličina formule u opštem slučaju nije u direktnoj vezi sa težinom problema) i da struktura formule bude pogodna za rešavanje. Stoga je uložen veliki napor da se razviju pogodna



Slika 2.9: Svođenje problema CSP na problem SAT.

prevođenja u problem SAT. Prevođenje koje na određen, uniforman način prevodi promenljive i ograničenja originalnog konačnog linearnog problema CSP u problem SAT zove se *SAT kodiranje* ili *kodiranje na SAT* (eng. *SAT encoding*). Za kodiranje ćemo reći da je *efikasno* ako je dobijena SAT formula pogodna za rešavanje od strane savremenih SAT rešavača (formula je "pogodna", ako je SAT rešavač brzo može rešiti). Ova formulacija nije precizna, jer formula pogodna za jedan rešavač ne mora biti pogodna za drugi. Jedan od ciljeva ove teze je i poboljšanje shvatanja o tome koja su kodiranja efikasna, odnosno kako prevesti promenljive i ograničenja problema CSP u pogodne SAT formule.

Treba naglasiti da se problem koji se rešava može i direktno modelirati pomoću iskazne formule u KNF-u i takav proslediti SAT rešavaču. Ovaj pristup ima i svoje nedostatke i prednosti. Nedostaci su da je za uspešnu primenu ovakvog pristupa potrebno dobro poznavanje mehanizma rada SAT rešavača i pogodnog načina izražavanja uslova CSP problema pomoću KNF formula, a to se ne može očekivati od prosečnog korisnika. Takođe, mogućnost greške se značajno povećava kada se modeliranje radi na ovako niskom nivou, tj. na nivou formule u KNF-u. Sa druge strane, pažljivim modeliranjem pomoću iskazne formule se može dobiti mnogo pogodnija

formula u KNF-u a samim tim i efikasnija metoda rešavanje problema.

Postoji nekoliko kodiranja koja su detaljno istraživana i opisana u literaturi ([132, 56, 125, 127, 55, 53]). Ova kodiranja se razlikuju po načinu na koji prevode celobrojne promenljive i ograničenja problema CSP u iskazne promenljive i klauze problema SAT. U poglavljima 2.3.3 do 2.3.6 dajemo prikaz najpoznatijih kodiranja. Svako kodiranje prevodi sve tipove ograničenja u SAT, a ovde se fokusiramo na neformalan opis kodiranja nekih jednostavnih tipova atomičnih ograničenja (na primer, ograničenja nejednakosti i jednakosti). Ideja je da se na jednostavnim primerima pokažu razlike u kodiranjima. Pre opisa kodiranja sledi opis jednog ograničenja nad iskaznim promenljivama koje se javlja kod raznih kodiranja pri svođenju problema CSP na SAT, kao i opisi svođenja problema CSP na neke probleme bliske problemu SAT.

Bulovska ograničenja kardinalnosti. Ograničenje nad iskaznim promenljivama $b_1, \ldots, b_n$ oblika:

$$b_1 + \ldots + b_n \ \# \ k, \quad k \in \mathbb{N}, \quad \# \in \{\leq, <, \geq, >, =\},$$

se zove *bulovsko ograničenje kardinalnosti* (eng. *boolean carinality constraint*). U prethodnih nekoliko godina razvijen je veliki broj različitih metoda za (efikasno) prevođenje ovog ograničenja u SAT (na primer, sekvencijalni i paralelni brojači [115], mreže kardinalnosti [7], uparene mreže kardinalnosti [36], kodiranja zasnovana na savršenom heširanju [14], itd.).

Bulovsko ograničenje kardinalnosti *tačno-jedan-tačan* $b_1 + \ldots + b_n = 1$ ($k = 1$, # je $=$) se najčešće javlja i označava da tačno jedna od promenljivih b_i mora uzeti vrednost 1. Jednostavan ali neefikasan način da se ovo ograničenje prevede u klauze je naivan način korišćenjem jedne *barem-jedan-tačan* klauze $b_1 \vee \ldots \vee b_n$ i kvadratnog broja *najviše-jedan-tačan* klauza $\neg b_i \vee \neg b_j$, gde je $1 \leq i < j \leq n$. Klauza oblika $\neg p \vee \neg q$ se zove *konfliktna* klauza[12] i ona onemogućava da dve iskazne promenljive p i q istovremeno budu tačne. Više efikasnijih načina za prevođenje *najviše-jedan-tačan* ograničenja od ovog jednostavnog načina je razvijeno (na primer, kodiranje proizvoda [32], komandant kodiranje [76]).

[12]Ovaj termin se koristi i za klauze prilikom procesa rešavanja iskazne formule, uvedene u poglavlju 2.3.1. Nadalje će se pod konfliktnom klauzom podrazumevati klauza oblika $\neg p \vee \neg q$.

Primer 12. *Da bi bulovsko ograničenje kardinalnosti* $b_1 + b_2 + b_3 = 2$ *bilo zadovoljeno, potrebno je da tačno dve od tri promenljive uzmu vrednost* 1*, na primer, da promenljive uzmu vrednosti* $b_1 = 1$*,* $b_2 = 1$ *i* $b_3 = 0$*.*

Mnogi tipovi ograničenja problema CSP se mogu lako prevesti u bulovska ograničenja kardinalnosti, a onda se ona nekom od mnogobrojnih metoda mogu prevesti u SAT formulu.

U opisima kodiranja koji slede, smatraćemo da x_i uzima vrednosti iz domena $D_i = [l_i, u_i]$ i x_j iz domena $D_j = [l_j, u_j]$. Svi opisi kodiranja su velikim delom preuzeti iz zajedničkog rada autora teze i Filipa Marića [122]. Treba spomenuti još dva kodiranja: kompaktno kodiranje uređenja [127] i logaritamsko kodiranje podrške [53], ali su ona u znatno manjoj upotrebi nego kodiranja koje opisujemo.

2.3.3 Direktno kodiranje

Direktno kodiranje razvijeno je još 80-ih godina prošlog veka ([40]) ali se često kao glavni rad vezan za njegov nastanak navodi rad Volša ([132]). Za svaku celobrojnu promenljivu x_i i svaku vrednost $v \in D_i$ uvodi se iskazna promenljiva $p_{i,v}$ koja uzima vrednost 1 akko je $x_i = v$. Tačno jedna od promenljivih $p_{i,v}$ mora uzeti vrednost 1, što se postiže uvođenjem bulovskog ograničenja kardinalnosti $p_{i,l_i} + \ldots + p_{i,u_i} = 1$. Na primer, ako x_1 uzima vrednosti iz domena $[3, 5]$, onda se uvodi vektor promenljivih $[p_{1,3}, p_{1,4}, p_{1,5}]$ i ograničenje $p_{i,3} + p_{i,4} + p_{i,5} = 1$. Dodela $[0, 1, 0]$ označava da je $x_1 = 4$.

Ako je potrebno kodirati ograničenje $x_i \neq x_j$, onda za svaku vrednost $v \in D_i \cap D_j$ treba da bude zadovoljena konfliktna klauza $\neg p_{i,v} \vee \neg p_{j,v}$.

Ako je potrebno kodirati ograničenje $x_i = x_j$, onda se za svake dve vrednosti $v \in D_i, w \in D_j$, takve da je $v \neq w$, uvodi klauza $\neg p_{i,v} \vee \neg p_{j,w}$. Malo bolje performanse se postižu ako se ove binarne konfliktne klauze uvedu samo za vrednosti $v, w \in D_i \cap D_j$, a jedinične klauze $\neg p_{i,v}$ i $\neg p_{j,w}$ se uvedu za sve vrednosti $v \in D_i \setminus D_j$ i $w \in D_j \setminus D_i$.

Ako je potrebno kodirati ograničenje $x_i < x_j$, onda se za svake dve vrednosti $v \in D_i, w \in D_j$, takve da je $w \leq v$, uvodi klauza $\neg p_{i,v} \vee \neg p_{j,w}$. Opet, malo bolje performanse se postižu ako se konfliktne klauze uvedu samo za vrednosti $v, w \in D_i \cap D_j$ (bez krajnjih tačaka) i ako se uvedu sledeće jedinične klauze: $\neg p_{i,v}$ gde $v \in D_i$ tako da je $w \leq v$ za sve $w \in D_j$, i $\neg p_{j,w}$ gde $w \in D_j$ tako da je $w \leq v$ za sve $v \in D_i$.

I za sve druge tipove ograničenja problema CSP, direktno kodiranje uglavnom koristi konfliktne klauze.

2.3.4 Kodiranje podrške

Prvi opis ovog kodiranja objavljen je još 1990. godine ([73]), ali je detaljniji opis ovog kodiranja zajedno sa njegovim karakteristikama pružio Gent ([56]). Kao i kod direktnog kodiranja uvode se iskazne promenljive $p_{i,v}$ i postavlja uslov da tačno jedna od njih uzima vrednost 1. Ipak, ograničenja se kodiraju na drugačiji način. Za svako ograničenje, umesto konfliktnih klauza, kodiranje podrške koristi *klauze podrške*. Ove klauze su negativne Hornove klauze ([56]), tj. sadrže najviše jedan negativan literal. Na primer, $p_{i,v_1} \rightarrow p_{j,v_2} \vee p_{j,v_3} \vee p_{j,v_4}$ je implikacija koja osigurava da ako x_i uzme vrednost v_1, onda x_j mora uzeti vrednost v_2, v_3 ili v_4 (vrednost v_1 iz domena x_i "podržava" vrednosti v_2, v_3 ili v_4 iz iz domena x_j). Prethodna implikacija je ekvivalentna klauzi podrške $\neg p_{i,v_1} \vee p_{j,v_2} \vee p_{j,v_3} \vee p_{j,v_4}$[13], a ova klauza se može tumačiti i na drugačiji način: ako x_j uzme jednu od vrednosti v_2, v_3 ili v_4, onda x_i može uzeti vrednost v_1, inače x_i ne može uzeti vrednost v_1. Mi ćemo u razmatranju kodiranja podrške uvek koristiti prvo tumačenje i samu implikaciju nazivati klauzom podrške.

Ako je potrebno kodirati ograničenje $x_i \neq x_j$, onda se za svaku vrednost $v \in D_i \cap D_j$ uvodi klauza podrške $p_{i,v} \rightarrow p_{j,l_j} \vee \ldots \vee p_{j,v-1} \vee p_{j,v+1} \vee \ldots \vee p_{j,u_j}$ ($x_i = v$ podržava sve vrednosti x_j različite od v). Simetrično, klauze koje označavaju da $x_j = w$ podržava sve vrednosti x_i različite od w se takođe uvode. Ne uvode se klauze za vrednosti van preseka $D_i \cap D_j$ pošto te vrednosti podržavaju sve vrednosti iz domena druge promenljive.

Ako je potrebno kodirati ograničenje $x_i = x_j$, onda se za svaku vrednost $v \in D_i \cap D_j$ uvode klauze dobijene prevođenjem $p_{i,v} \leftrightarrow p_{j,v}$ u KNF. Za iskazne promenljive $p_{i,v}$ i $p_{j,w}$ koje odgovaraju vrednostima $v \in D_i \setminus D_j$ i $w \in D_j \setminus D_i$, uvode se jedinične klauze $\neg p_{i,v}$ i $\neg p_{j,w}$.

Ako je potrebno kodirati ograničenje $x_i < x_j$, onda se za svaku vrednost $v \in D_i$ uvodi klauza $p_{i,v} \rightarrow p_{j,w_1} \vee \ldots \vee p_{j,w_m}$, gde je $W = \{w_1, \ldots, w_m\}$ skup svih vrednosti iz D_j većih od v. Ako je $W = D_j$, klauza ne mora da bude uvedena, a ako je $W = \emptyset$, onda klauza podrške postaje jedinična klauza $\neg p_{i,v}$. Simetrično, uvode se klauze koje označavaju da $x_j = w$ podržava sve vrednosti promenljive x_i manje od w.

Kodiranje linearne aritmetike za direktno kodiranje i kodiranje podrške. Za kodiranje ograničenja linearne aritmetike, koristi se sledeća shema (shema više

[13] Svaka implikacija $A \rightarrow B$ može se prevesti u klauzu $\neg A \vee B$.

odgovara kodiranju podrške, ali nisu sve generisane klauze negativne Hornove klauze).

Pretpostavimo da je x_k nova promenljiva i da je njen domen ograničen na vrednosti koje može da dobije izraz $x_k = ax_i$ ili $x_k = x_i + x_j$.

Ako je potrebno kodirati ograničenje $x_k = ax_i$, gde je a celobrojna konstanta, onda se uvode klauze $p_{i,v} \rightarrow p_{k,av}$. Na primer, ako je potrebno kodirati ograničenje $x_k = 2x_i$ pri čemu x_1 uzima vrednosti iz domena $[3, 5]$, onda promenljiva x_k ima domen $[6, 10]$ (odmah po definisanju promenljive x_k uvedene su i klauze koje zabranjuju da x_k uzme vrednosti 7 i 9, tj. $\neg p_{k,7}$ i $\neg p_{k,9}$) i uvode se klauze $p_{i,3} \rightarrow p_{k,6}$, $p_{i,4} \rightarrow p_{k,8}$ i $p_{i,5} \rightarrow p_{k,10}$. U opštem slučaju, moguće je uvesti i klauze $p_{k,u} \rightarrow p_{i,u/a}$, gde se u nalazi u domenu promenljive x_k, a to je u situaciji kada je u/a ceo broj.

Ako je potrebno kodirati ograničenje $x_k = x_i + x_j$, uvode se klauze $p_{i,v} \wedge p_{j,w} \rightarrow p_{k,v+w}$ ($a \wedge b \rightarrow c$ je isto što i klauza $\neg a \vee \neg b \vee c$) za svake dve vrednosti $v \in D_i, w \in D_j$. Na primer, ako je potrebno kodirati ograničenje $x_k = x_i + x_j$ pri čemu x_1 uzima vrednosti iz domena $[3, 4]$, a x_j vrednosti iz domena $[1, 2]$, onda promenljiva x_k ima domen $[4, 6]$ i uvode se klauze $p_{i,3} \wedge p_{j,1} \rightarrow p_{k,4}$, $p_{i,3} \wedge p_{j,2} \rightarrow p_{k,5}$, $p_{i,4} \wedge p_{j,1} \rightarrow p_{k,5}$ i $p_{i,4} \wedge p_{j,2} \rightarrow p_{k,6}$. U opštem slučaju, klauze $p_{k,u} \wedge p_{i,v} \rightarrow p_{j,u-v}$ i $p_{k,u} \wedge p_{j,w} \rightarrow p_{i,u-w}$ mogu biti uvedene.

2.3.5 Kodiranje uređenja

Kodiranje uređenja koje koristi sistem Sugar se pokazalo najefikasnijim za mnoge tipove problema, uključujući problem planiranja otvorene prodavnice [125], problem dvodimenzionalnog pakovanja trake [118] i generisanje test slučaja [9]. Razlozi efikasnosti leže u kompaktnosti ovog kodiranja i dobrim osobinama propagiranja [127]. Originalno kodiranje je opisano u radu Tamure i ostalih [125]. Ovde opisujemo malo drugačiju varijantu kodiranja uređenja (neformalno opisanu na veb-strani sistema Sugar) i dajemo dokaz korektnosti ovog kodiranja.

Celobrojna promenljiva x_i koja uzima vrednosti iz domena $D_i = [l_i, u_i]$ se predstavlja iskaznim promenljivama $p_{i,l_i}, \ldots, p_{i,u_i-1}$, gde $p_{i,v}$ uzima vrednost 1 akko je $x_i \leq v$. Jednostavnosti radi, promenljiva p_{i,l_i-1} koja je uvek netačna i promenljiva p_{i,u_i} koja je uvek tačna se takođe uvode. Veza između promenljivih je opisana pomoću klauza $p_{i,v-1} \rightarrow p_{i,v}$ (označavaju implikacije $x_i \leq v - 1 \rightarrow x_i \leq v$), uvedenih za svaku vrednost $v \in \{l_i, \ldots, u_i\}$.

Na slici 2.10 dat je pseudokod algoritma koji na jedan opšti način svodi ograničenja oblika $\sum_{k=1}^{n} a_k x_k \leq c$, gde su a_k i c celi brojevi i važi $a_k \neq 0$, na bulovsku

procedure kodiranje_uređenja
ulaz: ograničenje $\sum_{k=1}^{n} a_k x_k \leq c$, takvo da za sve k važi $a_k \neq 0$
izlaz: skup klauza nad atomima oblika $x_k \leq b_k$, za neke cele brojeve b_k
if $n = 1$ **then**
  if $a_1 > 0$ **then return** $\{x_1 \leq \lfloor c/a_1 \rfloor\}$
  if $a_1 < 0$ **then return** $\{\neg(x_1 \leq \lceil c/a_1 \rceil - 1)\}$
else
  izaberi neki x_i
  if $a_i > 0$ **then**
    return $\bigcup_{v \in D_i} \left(x_i \leq v - 1 \otimes \text{kodiranje\_uređenja } (\sum_{k=1,k\neq i}^{n} a_k x_k \leq c - a_i v)\right)$
  if $a_i < 0$ **then**
    return $\bigcup_{v \in D_i} \left(\neg(x_i \leq v) \otimes \text{kodiranje\_uređenja } (\sum_{k=1,k\neq i}^{n} a_k x_k \leq c - a_i v)\right)$

Slika 2.10: Procedura za kodiranje uređenja.

kombinaciju ograničenja oblika $x_k \leq b_k$, za neke cele brojeve b_k.

Operacija $l \otimes S$ označava dodavanje literala l svim klauzama u skupu S, tj., $l \otimes S = \{l \vee c.\ c \in S\}$.

Izbor promenljive x_i u proceduri je proizvoljan, ali je zbog efikasnosti pogodno izabrati x_i sa najmanjim domenom, gde se domeni po veličini porede pomoću broja dozvoljenih vrednosti, (i sa najmanjom apsolutnom vrednošću pridruženog koeficijenta u slučaju domena iste veličine), pošto takav izbor smanjuje veličinu generisane SAT instance.

Primer 13. *Opisaćemo kodiranje ograničenja* $3x_1 + 5x_2 \leq 14$*, gde je* $D_1 = [0, 5]$ *i* $D_2 = [0, 3]$*. Biramo promenljivu* x_2 *jer ima manji domen.*

Za $v_2 = 0$ *imamo* $x_2 \leq -1 \otimes 3x_1 \leq 14 - 5 \cdot 0$*, pa se uvodi klauza sa jednim literalom* $x_1 \leq 4$ *(*$x_2 \leq -1$ *nikada nije tačno).*

Za $v_2 = 1$ *imamo* $x_2 \leq 0 \otimes 3x_1 \leq 14 - 5 \cdot 1$*, pa se uvodi klauza* $x_2 \leq 0 \vee x_1 \leq 3$.

Za $v_2 = 2$ *imamo* $x_2 \leq 1 \otimes 3x_1 \leq 14 - 5 \cdot 2$*, pa se uvodi klauza* $x_2 \leq 1 \vee x_1 \leq 1$.

Za $v_2 = 3$ *imamo* $x_2 \leq 2 \otimes 3x_1 \leq 14 - 5 \cdot 3$*, pa se uvodi klauza* $x_2 \leq 2$.

Ograničenje $x_i \neq x_j$ je ekvivalentno sa $x_i - x_j \leq -1 \vee x_j - x_i \leq -1$. Ova dva linearna ograničenja se kodiraju odvojeno i njihova disjunkcija se kodira pomoću Ceitinove transformacije [131].

Ograničenje $x_i = x_j$ je ekvivalentno sa $x_i - x_j \leq 0 \wedge x_j - x_i \leq 0$. Ova dva linearna ograničenja se kodiraju odvojeno i jednostavno kombinuju.

Ograničenje $x_i < x_j$ je ekvivalentno linearnom ograničenju $x_i - x_j \leq -1$.

U nastavku dajemo dokaz korektnosti opisanog kodiranja uređenja koji nedostaje u literaturi. Završno prevođenje u iskazne promenljive (uvođenje iskaznih promenljivih $p_{i,v}$) je trivijalno.

Teorema 4. *Neka su x_i celobrojne promenljive sa domenima D_i, neka je C ograničenje $\sum_{k=1}^{n} a_k x_k \leq c$, gde su $a_k \neq 0$ i c celi brojevi, i neka je $S =$* kodiranje_uređenja *(C) skup klauza prvog reda koje su dobijene primenom procedure za kodiranje uređenja na C. U tom slučaju n-torka $(v_1, \ldots, v_n)$, gde $v_i \in D_i$, zadovoljava ograničenje C akko zadovoljava sve klauze u S.*

Tvrđenje se dokazuje indukcijom po n.

Ako je $n = 1$, onda je ograničenje C oblika $a_1 x_1 \leq c$.

Ako je $a_1 > 0$, onda je $S = \{x_1 \leq \lfloor c/a_1 \rfloor\}$. Važi da $\lfloor c/a_1 \rfloor \leq c/a_1$. Ako v_1 zadovoljava C, onda je $a_1 v_1 \leq c$, tj., $v_1 \leq c/a_1$, ali pošto je v_1 ceo broj takav da je $v_1 \leq c/a_1$, onda je $v_1 \leq \lfloor c/a_1 \rfloor$ pa v_1 zadovoljava S. Ako v_1 zadovoljava S, onda je $v_1 \leq \lfloor c/a_1 \rfloor \leq c/a_1$, pa $a_1 v_1 \leq c$, odnosno v_1 zadovoljava C.

Ako je $a_1 < 0$, onda je $S = \{\neg(x_1 \leq \lceil c/a_1 \rceil - 1)\} \equiv \{x_1 > \lceil c/a_1 \rceil - 1\}$. Ako v_1 zadovoljava C, onda je $a_1 v_1 \leq c$, pa je $v_1 \geq c/a_1$. Pošto važi da je $\lceil c/a_1 \rceil - 1 < c/a_1 \leq \lceil c/a_1 \rceil$, onda je $v_1 > \lceil c/a_1 \rceil - 1$, pa v_1 zadovoljava S. Ako v_1 zadovoljava S, onda je $v_1 > \lceil c/a_1 \rceil - 1$, pa pošto je v_1 ceo broj, onda je $v_1 \geq \lceil c/a_1 \rceil \geq c/a_1$, i v_1 zadovoljava C.

Pretpostavimo da važi induktivna hipoteza, tj. da je za sve $n' < n$ validno kodiranje svih ograničenja oblika $\sum_{k=1}^{n'} a'_k x_k \leq c'$, gde su $a'_k \neq 0$ i c' celi brojevi.

Ako je x_i fiksirano, neka za svako v u domenu x_i oznaka S_v označava skup klauza dobijenih iz ograničenja $C_v \equiv \sum_{k \neq i} a_k x_k \leq c - a_i v$ (tj., $S_v =$ kodiranje_uređenja (C_v)).

U prvom smeru predpostavljamo da $\mathbf{v} = (v_1, \ldots, v_n)$ zadovoljava C i sve domene promenljivih x_i i onda pokazujemo da $\mathbf{v}$ zadovoljava S. Važi da $\sum_{k=1}^{n} a_k v_k \leq c$. Neka je x_i izabrana promenljiva. Vrednost v_i je u domenu x_i.

Pretpostavimo da je $a_i > 0$. U tom slučaju, skup S se sastoji od klauza iz S_v, za svako v, proširenih literalom $x_i \leq v - 1$. Za svako $v \geq v_i + 1$, izraz $v_i \leq v - 1$ je tačan, pa su sve klauze u S dobijene iz S_v zadovoljene. Ako je $v \leq v_i$, onda je $c \geq \sum_{k \neq i} a_k x_k + a_i v_i \geq \sum_{k \neq i} a_k x_k + a_i v$, pa n-torka dobijena iz $\mathbf{v}$ uklanjanjem v_i zadovoljava C_v. Po induktivnoj hipotezi ova n-torka zadovoljava sve klauze iz S_v i one ostaju zadovoljene kada se prošire literalom $x_i \leq v - 1$.

Pretpostavimo da je $a_i < 0$. U tom slučaju, skup S se sastoji od klauza iz S_v, za svako v, proširenih literalom $\neg(x_i \leq v)$. Za svako $v < v_i$, izraz $\neg(v_i \leq v)$ je tačan, pa su sve klauze iz S dobijene iz S_v zadovoljene. Ako je $v \geq v_i$, onda je $c \geq \sum_{k \neq i} a_k x_k + a_i v_i \geq \sum_{k \neq i} a_k x_k + a_i v$ (pošto je $a_i < 0$), pa n-torka dobijen iz $\mathbf{v}$ uklanjanjem v_i zadovoljava C_v. Po induktivnoj hipotezi ova n-torka zadovoljava sve klauze iz S_v i one ostaju zadovoljene kada se prošire literalom $\neg(x_i \leq v)$.

U suprotnom smeru, pretpostavljamo da n-torka $\mathbf{v}$ zadovoljava S i sve domene promenljivih x_k i pokazujemo da ova n-torka zadovoljava C. Neka je x_i izabrana promenljiva. U tom slučaju, v_i je u domenu x_i.

Pretpostavimo da je $a_i > 0$. Dovoljno je razmotriti samo klauze dobijene iz S_v, za $v = v_i$. Ovo su klauze iz S_{v_i}, proširene literalom $x_i \leq v_i - 1$. Ali izraz $v_i \leq v_i - 1$ je netačan, pa sve klauze iz S_{v_i} moraju biti zadovoljene n-torkom dobijenom iz $\mathbf{v}$ uklanjanjem v_i. Po induktivnoj hipotezi, ograničenje $C_v \equiv \sum_{k \neq i} a_k x_k \leq c - a_i v_i$ je zadovoljeno istom n-torkom, pa $\mathbf{v}$ takođe zadovoljava C.

Pretpostavimo da je $a_i < 0$. Kao i u prethodnom slučaju, dovoljno je razmotriti samo klauze dobijene iz S_v, za $v = v_i$. Ovo su klauze iz S_{v_i}, proširene literalom $\neg(x_i \leq v_i)$. Ali izraz $\neg(v_i \leq v_i)$ je netačan, pa sve klauze iz S_{v_i} moraju biti zadovoljene n-torkom dobijenom iz $\mathbf{v}$ uklanjanjme v_i. Po induktivnoj hipotezi, ograničenje $C_v \equiv \sum_{k \neq i} a_k x_k \leq c - a_i v_i$ je zadovoljeno istom n-torkom, pa $\mathbf{v}$ takođe zadovoljava C.

2.3.6 Logaritamsko kodiranje

Osnovno logaritamsko kodiranje razvijeno je još 90-ih godina prošlog veka ([68]), a kasnije je ono detaljnije opisano i unapređeno ([55, 132, 53]). Ovo kodiranje odgovara predstavljanju brojeva u računaru. Svaka celobrojna promenljiva se kodira sa istim brojem n iskaznih promenljivih (tj. bitova). Broj n se bira tako da maksimalna i minimalna vrednost u domenu svih promenljivih mogu biti predstavljene (na primer, 32, što odgovara standardnoj širini mašinske reči). Za svaku celobrojnu promenljivu x_i, iskazna promenljiva $p_{i,k}$ je tačna akko je k-ti bit binarne reprezentacije vrednosti dodeljene x_i jednak 1. Za promenljive koje uzimaju negativne vrednosti koristi se reprezentacija komplementa dvojke (negativne vrednosti se retko javljaju kod problema CSP pa je opis logaritamskog kodiranja obično ograničen na promenljive nenegativnih domena). Ako je gornja granica nenegativne promenljive značajno manja od 2^n, onda se određeni broj bitova većeg značaja postavlja na

0. Ako gornja granica nenegativne promenljive nije stepen dvojke, nedozvoljene vrednosti se moraju isključiti. Za svaku takvu vredost v koja nije u domenu x_i i predstavljena je pomoću binarnih cifara $v_{n-1}, \ldots, v_0$ ($v = \sum_{k=0}^{n-1} 2^k v_k$), uvodi se disjunkcija $\bigvee_{k=0}^{n-1} v_k \oplus p_{i,k}$. Simbol $\oplus$ označava ekskluzivnu disjunkciju i $v_k \oplus p_{i,k}$ dobija vrednost 1 ako ova dva bita uzimaju različite vrednosti, a dobija vrednost 0 ako uzimaju iste vrednosti.

Ako je potrebno kodirati ograničenje $x_i \neq x_j$, onda se uvodi disjunkcija $\bigvee_{k=0}^{n-1} p_{i,k} \oplus p_{j,k}$.

Ako je potrebno kodirati ograničenje $x_i = x_j$, onda se generišu klauze dobijene prevođenjem $p_{i,k} \leftrightarrow p_{j,k}$ u KNF, za svaki indeks k.

Ako je potrebno kodirati ograničenje $x_i < x_j$, onda se uvodi skup promenljivih $\{d_0, \ldots, d_{n-1}\}$, gde je $d_k = 1$ akko je ograničenje zadovoljeno uzimajući u obzir samo bitove sa indeksima $0, \ldots, k$ promenljivih x_i i x_j. Sledeće klauze se kodiraju: $d_0 \leftrightarrow \neg p_{i,0} \wedge p_{j,0}$, $d_k \leftrightarrow (d_{k-1} \wedge \neg p_{i,k} \wedge \neg p_{j,k}) \vee (d_{k-1} \wedge p_{i,k} \wedge p_{j,k}) \vee (\neg p_{i,k} \wedge p_{j,k})$, gde je k indeks bita i uzima vrednosti od 1 do $n-2$, i $d_{n-1} \leftrightarrow (d_{n-2} \wedge \neg p_{i,n-1} \wedge \neg p_{j,n-1}) \vee (d_{n-2} \wedge p_{i,n-1} \wedge p_{j,n-1}) \vee (p_{i,k} \wedge \neg p_{j,k})$. Jedinična klauza d_{n-1} se takođe generiše.

Aritmetičke operacije se kodiraju pomoću formula koje opisuju sabiranje brojeva u komplementu dvojke, i ovde nećemo navoditi klauze koje se u tom slučaju generišu.

2.3.7 Metode rešavanja bliske svođenju na SAT

Opisaćemo ukratko nekoliko problema koji su bliski problemu SAT i koji se takođe koriste da bi se rešili problemi CSP.

2.3.7.1 Problem PB i Partial MaxSAT problem

Ograničenja nad iskaznim promenljivama $b_1, \ldots, b_n$ oblika:

$$c_1 b_1 + \ldots + c_n b_n \ \geq c_{n+1}, \quad c_1, \ldots, c_{n+1} \in \mathbb{Z},$$

se zovu *pseudo-bulovska* ograničenja (eng. *pseudo-boolean*), skraćeno PB [47]. Ova ograničenja se mogu smatrati generalizacijom klauza, tj. u specijalnom slučaju kada je za sve vrednosti c_i ($i = 1, \ldots, n+1$) ispunjeno $c_i = 1$, PB ograničenje postaje klauza. Ova ograničenja se mogu smatrati i generalizacijom bulovskih ograničenja kardinalnosti, tj. u specijalnom slučaju kada je za sve vrednosti c_i ($i = 1, \ldots, n+1$) ispunjeno $c_i \geq 0$, PB ograničenje postaje bulovsko ograničenje kardinalnosti. Primetimo da korišćenje samo aritmetičke relacije $\geq$ u definiciji nije ograničavajuće,

pošto se ostale aritmetičke relacije mogu predstaviti pomoću jednog ili više PB ograničenja (na primer, relacija $\leq$ se može svesti na $\geq$ istovremenim prebacivanjem izraza sa leve strane na desnu i izraza sa desne strane na levu).

Pseudo-bulovski optimizacioni problem je problem određivanja maksimuma ili minimuma zadatog linearnog izraza nad iskaznim promenljivama $b_1, \ldots, b_n$:

$$c_1 b_1 + \ldots + c_n b_n, \quad c_1, \ldots, c_n \in \mathbb{Z},$$

pri čemu moraju da budu zadovoljena sva zadata PB ograničenja. Ovaj problem se često naziva i 0-1 problem celobrojnog linearnog programiranja, jer to jeste problem celobrojnog linearnog programiranja sa restrikcijom domena. Kao i u slučaju problema SAT, danas postoji veliki broj javno dostupnih, slobodnih i besplatnih rešavača koji rešavaju PB problem u standarizovanom formatu PBS[14], a održavaju se i takmičenja ovih rešavača. Mnogi realni problemi se mogu rešiti modeliranjem problema kao problema PB i potom rešavanjem pomoću savremenih PB rešavača, koji obično koriste jedan od dva pristupa. U prvom pristupu se koristi svođenje PB ograničenja na klauze i veliki broj radova na ovu temu postoji (na primer, [35, 49, 47, 119]). Do poboljšanja performansi rešavača se obično dolazi i direktnom obradom ograničenja pre svođenja na klauze (na primer, [1]). U drugom pristupu se SAT rešavač proširuje da bi mogao da pored klauza obrađuje i PB ograničenja ([16]).

Partial MaxSAT problem [19] je optimizaciona verzija problema SAT u kojoj je potrebno naći vrednosti iskaznih promenljivih tako da su zadovoljena sve *tvrde klauze* (eng. *hard clauses*) i da je maksimizovan broj zadovljenih *mekih klauza* (eng. *soft clauses*). I za ovaj problem postoje standardizovani format nazvan WCNF[15], veliki broj javno dostupnih rešavača, a takmičenja Partial MaxSAT rešavača se redovno organizuju.

2.3.7.2 Svođenje na SMT

Satisfiability modulo theories, skraćeno SMT [19], je oblast koja se bavi ispitivanjem zadovoljivosti (rešavanjem) formule u nekoj odlučivoj teoriji prvog reda (ili kombinaciji teorija). Neke od ovih teorija su *linearna celobrojna aritmetika* (LIA), *linearna realna aritmetika* (LRA), *teorija bit vektora* (BV), *Logika razlika* (DL),

[14] `http://www.cril.univ-artois.fr/PB12/format.pdf`
[15] `http://maxsat.ia.udl.cat/requirements`

Teorija nizova (A), *teorija neinterpretiranih funkcija* (UF), itd. Programi za ispitivanje zadovoljivosti SMT formula se zovu *SMT rešavači* (eng. *SMT solvers*), a standardizovani ulazni format ovih rešavača je SMT-LIB[16]. Na slici 2.11 prikazan je problem 4 kraljice koji je predstavljen u teoriji linearne celobrojne aritmetike.

```
; Komentari počinju znakom ';'
(set-logic QF_LIA) ; oznaka teorije
(set-info :smt-lib-version 2.0)

(declare-fun x1 () Int) ; deklarisanje konstante
(assert (>= x1 1))
(assert (<= x1 4))
... ; na analogan način deklarisane su x2 x3 x4
(assert (distinct x1 x2 x3 x4 ))

(declare-fun s1 () Int)
(assert (>= s1 2))
(assert (<= s1 5))
(assert (= s1 (+ x1 1 )))
... ; na analogan način deklarisane su s2 s3 s4
(assert (distinct s1 s2 s3 s4)) ; vrednosti s1, s2, s3 i s4 su različite

(declare-fun s5 () Int)
(assert (>= s5 0))
(assert (<= s5 3))
(assert (= s5 (+ x1 (- 1) )))
... ; na analogan način deklarisane su s6 s7 s8
(assert (distinct s5 s6 s7 s8))

(check-sat) ; oznaka traženja rešenja
(get-value (x1)) ; dobijanje vrednosti x1
(get-value (x2))
(get-value (x3))
(get-value (x4))
(exit)
```

Slika 2.11: Problem 4 kraljice u SMT-LIB 2 jeziku.

Jedan pristup rešavanju SMT formula je prevođenje tih formula u SAT formulu. Prevođenje u SAT formulu se vrši u nizu koraka (na primer, potrebno je prvo suziti domene celobrojnih promenljivih) što še obično obavlja u toku preprocesiranja SMT formule koju treba rešiti. Za ovakav pristup rešavanju se kaže da je *vredan*

[16] `http://www.smtlib.org`

(eng. *eager*) i njegova prednost je što mogu da se iskoriste sve efikasniji SAT rešavači za rešavanje formule, kako ovi rešavači postaju dostupni. Ovaj pristup ima nekoliko nedostataka ([59]). Prvo, prevođenjem u SAT formulu se gubi struktura SMT formule i mnoge informacije o problemu. Drugo, uprkos preprocesiranju, generisana SAT formula može biti ogromna i time praktično nerešiva pomoću savremenih SAT rešavača. Treće, ovakav pristup se ne prilagođava lako kombinovanju različitih teorija. Navedeni pristup se danas skoro uopšte više ne koristi.

Da bi se prevazišli pomenuti nedostaci, razvijen je *lenji* (eng. *lazy*) pristup i većina savremenih SMT rešavača koriste ovaj pristup. Pri implementaciji SMT rešavača najčešće se koristi DPLL($\mathcal{T}$) arhitektura [99] ($\mathcal{T}$ je teorija prvog reda). Ova arhitektura se zasniva na eksplicitnom razdvajanju SAT rešavača od rešavača za pojedine teorije. SAT rešavač i rešavač teorije komuniciraju i razmenjuju informacije do kojih dođu. SAT rešavač je zadužen za iskaznu strukturu formule a rešavač teorije za delove koji se odnose na samu teoriju. Dva rešavača komuniciraju kroz striktno definisani interfejs, a SMT rešavač se može proširiti dodavanjem rešavača i za druge teorije, pod uslovom da se novi rešavači uklapaju u arhitekturu i implementiraju odgovarajući interfejs. Prednost korišćenja rešavača teorije je što oni na raspolaganju imaju algoritme i strukture podataka pogodne za odgovarajuće teorije. Sa druge strane, SAT rešavači su odlični u rešavanju iskaznih formula.

SMT rešavači koji implementiraju lenji pristup koriste skup pravila sličan onome koji je opisan u poglavlju o SAT rešavačima (2.3.1.3). Pojedina pravila se koriste u istom obliku jer ne zavise od teorije (`Decide`, `Unit Propagate`, `Restart`) a ostala pravila se prilagođavaju tako da zavise od konkretne teorije. Prilikom izgradnje valuacije, SAT rešavač povremeno traži od rešavača teorije da proveri da li je ta valuacija neprotivrečna u toj teoriji. Ukoliko jeste, nastavlja se sa proširivanjem valuacije. Ukoliko nije, uči se konfliktna klauza (odgovara pasusu Analiza konflikta poglavlja 2.3.1.3 kod SAT rešavača) koja za cilj ima da se u budućnosti spreči izgradnja iste valuacije.

2.3.7.3 Lenjo generisanje klauza

Primer 14 (Motivacioni primer)**.** *Razmotrimo problem CSP u kome su definisane tri celobrojne promenljive x_1, x_2, x_3 koje imaju isti domen $[-1000, 1000]$ i ograničenje $x_1 + x_2 = x_3$ (može postojati još promenljivih i ograničenja u problemu ali mi razmatramo samo navedene promenljive i ograničenje). Pretpostavimo da je kodiranje uređenja korišćeno za svođenje na SAT, tj. za promenljivu x_i ($i \in \{1, 2, 3\}$)*

uvedene su iskazne promenljive $x_{i,-1001}, \ldots, x_{i,1000}$, gde $x_{i,p}$ označava da je $x_i \leq p$, a ograničenje je kodirano na način opisan u poglavlju 2.3.5. Ovim kodiranjem uvode se hiljade promenljivih i milioni klauza i problem zbog svoje veličine postaje potencijalno težak za savremene SAT rešavače. Moguće rešenje problema leži u uvođenju promenljivih i klauza samo kada one zaista i budu potrebne (na primer, kada se po promenljivoj vrši grananje ili kada se propagiranjem jediničnih klauza utvrdi vrednost neke promenljive). Da bi se ovo postiglo, potrebno je da se određeni zadaci izvršavaju van SAT rešavanja, tj. potrebno je uvesti poseban mehanizam koji će sarađivati sa SAT rešavačem.

Lenjo generisanje klauza (eng. *lazy clause generation*) [103, 50] kombinuje propagiranje ograničenja sa SAT rešavanjem – naizmenično se koriste propagiranje ograničenja i SAT rešavač i međusobno razmenjuju zaključke do kojih dolaze. Postoje dva pristupa za lenjo generisanje klauza i kod oba pristupa se literali i klauze ne generišu odmah na početku, već se dinamički konstruišu tokom izvršavanja, u trenutku kada postanu potrebni.

U prvom, originalnom pristupu [103] pretragu kontroliše SAT rešavač. Propagiranjem jediničnih klauza se sužavaju domeni promenljivih (promenljive i njihovi domeni su deo mehanizma koji vrši propagiranje ograničenja). Kada se propagiranje jediničnih klauza završi, pokreće se propagiranje ograničenja i pri tome se za svako smanjivanje domena promenljivih generiše klauza koja objašnjava ovo smanjivanje i ta klauza se prosleđuje SAT rešavaču. Kada se propagiranje ograničenja završi, ponovo se pokreće propagiranje jediničih klauza. Dve vrste propagiranja se naizmenično ponavljaju do trenutka kada se domen nijedne promenljive ne može smanjiti, tj. do trenutka kada mora da se donese odluka (izvrši grananje). Pored propagiranja jediničnih klauza koriste se i svi ostali mehanizmi za efikasan rad SAT rešavača.

U drugom pristupu [50] pretragu kontroliše CSP rešavač. Kada se smanji domen neke promenljive, klauza koja objašnjava smanjenje se šalje SAT rešavaču a u listu operacija koje treba dodati se stavlja propagiranje jediničnih klauza SAT rešavača (ono je najvišeg prioriteta) i operacije propagiranja ograničenja koje se odnose na tu promenljivu. Svako propagiranje koje smanji domen neke promenljive utiče na stavljanje drugih propagiranja u listu operacija, pri čemu se propagiranje jediničnih klauza dodaje pri bilo kojoj promeni domena.

Lenjo generisanje klauza se može posmatrati kao specijalan slučaj SMT rešavanja, gde se propagiranje ograničenja posmatra kao zaključivanje u pozadinskoj teoriji. Samim tim, lenjo generisanje klauza je mnogo pogodnije za rešavanje pro-

blema koji uključuju ograničenja.

2.3.8 Sistemi koji koriste svođenje na SAT i bliske metode

Ovde navodimo postojeće sisteme za svođenje problema CSP na problem SAT, kao i na srodne probleme.

Spec2SAT je aplikacija koja omogućava prevođenje NPSpec specifikacije (kada je data zajedno sa ulaznim podacima) u SAT instance.

FznTini [62] uvodi prevođenje bilo kog problema zadovoljivosti ili optimizacije u FlatZinc jeziku (koji ne sadrži realne brojeve) u SAT formulu koja se onda rešava pomoću jednog ili više poziva SAT rešavača.

Alatka `fzn2smt` [24] prevodi FlatZinc u jezik SMT-LIB.

Rešavači `mzn-g12cpx` (poznat i pod nazivom `Opturion CPX`) i `mzn-g12lazy` koriste lenjo generisanje klauza i dostupni su u MiniZinc distribuciji.

ursa familija alatki (ursa, URBiVA, ursa major) [69, 91] uvodi uniformno svođenje specifikacija u jeziku sličnom jeziku C na SAT i različite SMT teorije. Prevođenje ima preciznu semantiku, komunikacija sa SAT/SMT rešavačima se vrši pomoću njihovih API-ja i nalaženje svih rešenja je podržano.

Sugar rešava konačne linearne probleme CSP prevođenjem u SAT korišćenjem kodiranja uređenja [125] i onda rešavanjem SAT instanci pomoću nekoliko dostupnih SAT rešavača.

Azucar [128] je naslenik sistema Sugar koji koristi kompaktno kodiranje uređenja [127] za prevođenje konačnih linearnih problema CSP u SAT i podešen je za rešavanje CSP instanci sa velikim domenima promenljivih.

bee [94] (Ben-Gurion University Equi-propagation Encoder) je kompajler za prevođenje probleme ograničenja sa konačnim domenima u KNF korišćenjem kodiranja uređenja [125], ali uz određene optimizacije.

Sat4j [16] je biblioteka za rešavanje problema CSP, SAT, PB i još nekoliko bliskih problema problemu SAT. Razvijen je i istoimeni sistem koji između ostalog omogućava rešavanje problema CSP svođenjem na SAT korišćenjem direktnog kodiranja i kodiranja podrške.

2.4 Predlog novih kodiranja CSP na SAT

U ovom poglavlju ćemo predstaviti dva nova, hibridna kodiranja koja predstavljaju kombinacije već postojećih kodiranja.

2.4.1 Kodiranje direktno-podrška

Celobrojne promenljive se kod direktnog kodiranja i kodiranja podrške na isti način prevode u bulovsku reprezentaciju. Ali svako od ovih kodiranja prevodi neka od ograničenja u klauze koje sadrže manje literala od drugog kodiranja. Predlažemo kombinaciju ova dva kodiranja, tj. hibridno kodiranje koje je zasnovano na procenjenoj veličini kodiranja odgovarajućih ograničenja. Celobrojne promenljive se kodiraju na isti način kao kod oba navedena kodiranja. Ograničenja se u nekim slučajevima kodiraju pomoću direktnog kodiranja, u nekim slučajevima pomoću kodiranja podrške, a u nekim slučajevima se ograničenje delom kodira jednim a delom drugim kodiranjem. Priroda ograničenja određuje koje kodiranje će se upotrebiti kao deo hibridnog: direktno kodiranje je pogodnije kada se kodiraju konfliktna ograničenja, a kodiranje podrške je pogodnije kada se kodiraju ograničenja podrške.

Ako je potrebno kodirati ograničenje $x_i \neq x_j$, onda je broj klauza koje se generišu u kodiranju podrške približno dvostruko veći od odgovarajućeg broja kod direktnog kodiranja. Dodatno, kod direktnog kodiranja se klauze uvek sastoje od dva literala, dok se kod kodiranja podrške klauze sastoje od dva ili više literala. Zbog ovih razloga, kodiranje direktno-podrška koristi klauze direktnog kodiranja za ovo ograničenje.

Ako je potrebno kodirati ograničenje $x_i = x_j$, onda direktno kodiranje generiše $O(|D_i| \cdot |D_j|)$ klauza, a kodiranje podrške $O(\max\{|D_i|, |D_j|\}))$ klauza. U oba slučaja, sve generisane klauze imaju po dva literala. Kodiranje direktno-podrška koristi klauze kodiranja podrške za ovo ograničenje, jer je manje veličine.

Ako je potrebno kodirati ograničenje $x_i < x_j$, onda se za svaku vrednost $v \in D_i$ biraju ili klauze direktnog kodiranja ili klauze kodiranja podrške, sa ciljem da se minimizuje broj literala. Ako je v u konfliktu sa najviše trećinom vrednosti iz D_j, onda se koriste klauze direktnog kodiranja. U suprotnom, v podržava najviše dve trećine vrednosti iz D_j i koriste se klauze iz kodiranja podrške.

2.4.2 Kodiranje direktno-uređenje

Nedostatak kodiranja uređenja je što ono nije pogodno za kodiranje nekih globalnih ograničenja. Novija istraživanja [94, 103] su pokazala da se neki problemi najefikasnije rešavaju kada se kodiranje uređenja kombinuje sa direktnim kodiranjem ograničenja *all-different*. U tim istraživanjima promenljive koje ograničava *all-different* se kodiraju pomoću oba kodiranja, i klauze koje povezuju dve reprezentacije (*klauze povezivanja*) se generišu.

Vršimo nadgradnju postojećih istraživanja i uvodimo kodiranje direktno-uređenje (eng. *direct-order*)[17]. Ovo kodiranje koristi direktno kodiranje specijalnih slučajeva globalnih ograničenja koja su opisana u narednom pasusu, a za sve druge tipove ograničenja se koristi kodiranje uređenja. Sve celobrojne promenljive kodiraju se pomoću kodiranja uređenja. Pošto se globalna ograničenja zasnivaju na direktnom kodiranju, za celobrojne promenljive na kojima su definisana ova ograničenja uvode se i iskazne promenljive i klauze koje odgovaraju direktnom kodiranju. Iskazne promenljive direktnog kodiranja i kodiranja uređenja se povezuju klauzama povezivanja. Pretpostavimo da za celobrojnu promenljivu x_i kodiranje uređenja uvodi iskazne promenljive $p_{i,v}$ a direktno kodiranje $p'_{i,v}$. Klauze koje povezuju ova dva skupa promenljivih se dobijaju iz: $p'_{i,v} \leftrightarrow \neg p_{i,v-1} \wedge p_{i,v}$, za svaku vrednost $v \in D_i$. Primetimo da se klauze koje u direktnom kodiranju zabranjuju mogućnost da celobrojna promenljiva uzima više vrednosti mogu izostaviti, pošto kodiranje uređenja obezbeđuje jedinstvenu vrednost promenljive.

Za pojedina globalna ograničenja (na primer, ograničenje *count*) se u nekim specijalnim slučajevima može uvesti efikasno direktno kodiranje i te specijalne slučajeve ćemo zvati *specijalizovana globalna ograničenja* (na primer, specijalan slučaj ograničenja *count*). U ostalim slučajevima je vrlo teško uvesti efikasno direktno kodiranje i te slučajeve ćemo zvati *nespecijalizovana globalna ograničenja*. Globalna ograničenja kod kojih ne postoji specijalan slučaj (ili mi nismo svesni njegovog postojanja) zovemo *generalna globalna ograničenja* (na primer, sva pojavljivanja ograničenja poput *lex_less*, *element*, itd.).

[17] Ovo kodiranje se može nazvati i podrška-uređenje ili direktno-podrška-uređenje, pošto se (globalna) ograničenja koja se u ovom slučaju kodiraju pomoću direktnog kodiranja na isti način kodiraju pomoću kodiranja podrške.

2.4.2.1 Ograničenje *count*

Ograničenje *count* $(e_1, \ldots, e_n, v)$ *op* c, opisano u poglavlju 2.1.2 se obično kodira svođenjem na jednostavnija ograničenja:

$$(\mathit{if}\ e_1 = v\ \mathit{then}\ 1\ \mathit{else}\ 0) + ... + (\mathit{if}\ e_n = v\ \mathit{then}\ 1\ \mathit{else}\ 0)\ \mathit{op}\ c,$$

a ovo ograničenje se prevodi u ograničenja $s_1 + \ldots + s_n\ \mathit{op}\ c$ i $((e_i = v) \rightarrow (s_i = 1)) \wedge ((e_i \neq v) \rightarrow (s_i = 0))$, $i \in \{1, \ldots, n\}$, gde su $s_1, \ldots, s_n$ novouvedene promenljive.

Specijalizovano ograničenje *count*. Pri modeliranju problema CSP obično se koristi specijalan slučaj ovog ograničenja: i v i c su konstante (a ne izrazi kao što je dozvoljeno u opštem slučaju), a $e_1, \ldots, e_n$ su promenljive. Ovaj specijalan slučaj može biti preveden u SAT pomoću direktnog kodiranja na posebno prilagođen način. Jednostavnosti radi, pretpostavimo da su domeni svih promenljivih $[l, u]$. Ove promenljive su predstavljene pomoću iskaznih promenljivih (svaki red predstavlja jednu celobrojnu promenljivu):

$$\begin{matrix} p_{1,l} & \cdots & p_{1,u} \\ \vdots & \ddots & \vdots \\ p_{n,l} & \cdots & p_{n,u} \end{matrix}$$

Specijalan slučaj ograničenja *count* može se na jednostavan način kodirati u SAT uvođenjem bulovskih ograničenja kardinalnosti $p_{1,v} + p_{2,v} + \ldots + p_{n,v}\ \#\ c$ (ukoliko domeni promenljivih nisu isti, onda se za svaku celobrojnu promenljivu x_i takvu da v nije u njenom domenu, iskazna promenljiva $p_{i,v}$ isključuje iz sume).

Kodiranje uređenja ne koristi bulovska ograničenja kardinalnosti i zato ne može koristiti ovo specijalno prevođenje (čak i kada bi se koristila bulovska ograničenja kardinalnosti prevođenje ne bi bilo jednostavno kao u slučaju direktnog kodiranja).

Nespecijalizovano ograničenje *count*. U slučaju da bilo v ili c nije konstanta, ne može koristiti opisano svođenje na bulovska ograničenja kardinalnosti pošto ova ograničenja zahtevaju konkretnu vrednost za v da bi bile određene promenljive na kojima se zadaje ovo ograničenje i konkretnu vrednost c da bi se znalo koliko iskaznih promenljivih mora biti tačno. Retki su problemi gde bilo v ili c nije konstanta.

2.4.2.2 Ostala globalna ograničenja

Ostala globalna ograničenja takođe imaju specijalne slučajeve koji se obično koriste pri modeliranju problema CSP. Od 11 globalnih ograničenja podržanih Sugar jezikom i opšti oblik ograničenja *all-different* i specijalizovani slučajevi 4 druga ograničenja (*global_ cardinality*, *global_ cardinality_ with_ costs*, *nvalue*, *cumulative*) mogu efikasno da se kodiraju pomoću direktnog kodiranja. Mnoga druga ograničenja iz kataloga globalnih ograničenja se takođe mogu kodirati na specijalan način pomoću direktnog kodiranja. U opštem slučaju, direktno kodiranje može efikasno kodirati globalna ograničenja u slučajevima kada se ova ograničenja mogu postaviti pomoću bulovskih ograničenja kardinalnosti na iskaznim promenljivama različitih celobrojnih promenljivih koje odgovaraju istoj vrednosti.

Mnoga od ovih globalnih ograničenja mogu da se svedu na specijalan slučaj ograničenja *count*, pa se unapređivanjem obrade ovog ograničenja i ostala ograničenja efikasnije obrađuju. Na primer, primetimo da se kodiranje ograničenja *all-different* može svesti na kodiranje specijalnog slučaja ograničenja *count*. Naime, za svaku vrednost iz unije domena argumenata ograničenja *all-different*, može se uvesti ograničenje *count* koje ograničava da broj pojavljivanja te vrednosti u skupu izraza koji su argumenti ograničenja *all-different* mora biti manji ili jednak od 1.

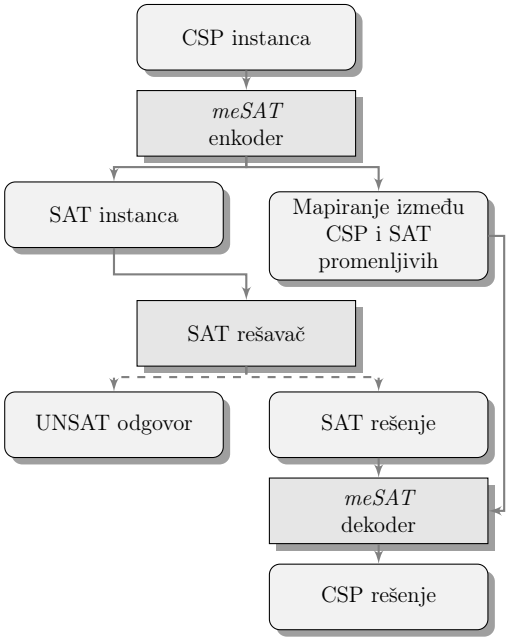
2.5 Implementacija sistema meSAT i njegova eksperimentalna evaluacija

U ovom poglavlju ćemo prvo opisati sistem *meSAT* koji smo razvili. Potom ćemo prikazati rezultate eksperimenata koji opravdavaju uvođenje opisanih hibridnih kodiranja. Na kraju, izvršićemo poređenje sistema *meSAT* sa već postojećim sistemima za svođenje na SAT.

2.5.1 Sistem *meSAT*

Sistem *meSAT*[18] [122] je implementiran u jeziku C++. Već postoji veliki broj jezika za modeliranje problema CSP (oni su opisani u poglavlju 2.1.6). Stoga, nismo želeli da razvijamo novi jezik, i *meSAT* trenutno podržava jezik sistema Sugar. Ovaj

[18]Izvorni kod sa primerima je dostupan na adresi: `http://argo.matf.bg.ac.rs`

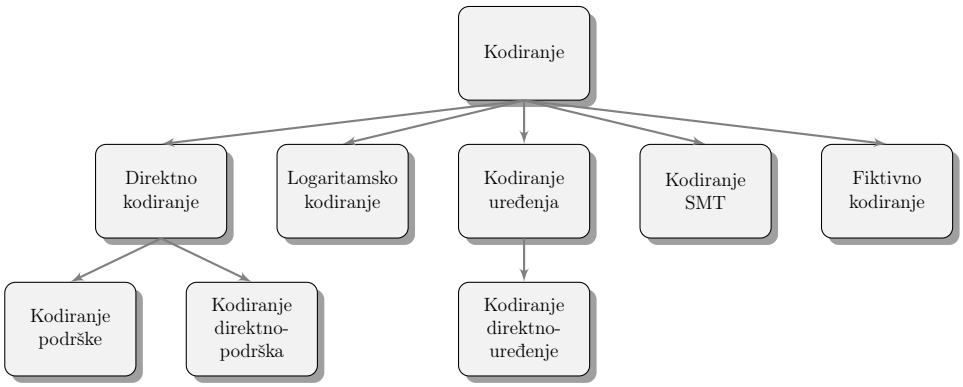


Slika 2.12: Arhitektura sistema *meSAT*.

jezik je izabran jer je niskog nivoa i sve konstrukcije se relativno jednostavno mogu prevesti u SAT.

Postupak rešavanja problema CSP svođenjem na SAT odgovara postupku već opisanom u Poglavlju 2.3.2 i prikazan je na slici 2.12. Svođenje na SMT se obavlja po istom principu. Pri svođenju na SAT, ili se može samo generisati izlazna DIMACS datoteka, ili se dodatno može pokrenuti jedan od podržanih SAT rešavača: MiniSat [45], *clasp* [54] ili Lingeling [18] – u slučaju da je problem zadovoljiv, valuacija se prevodi u rešenje problema CSP. Analogno, pri svođenju na SMT, može se ili samo generisati SMT-LIB instanca, ili se dodatno može pokrenuti jedan od podržanih SMT rešavača: Yices [44] ili Z3 [41]. U slučaju rešavanja problema COP, rešavač se iznova pokreće za različite vrednosti celobrojne promenljive čiju najveću/najmanju vrednost treba odrediti, sve dok se ne pronađe ta (optimalna) vrednost. Sledeća vrednost se bira pomoću jednostavnog algoritma binarne pretrage. COP instanca se smatra rešenom, samo ako je optimalna vrednost određena i ako je njena optimalnost pokazana u okviru datog vremenskog ograničenja.

Sistem *meSAT* ima fleksibilnu arhitekturu. Ulazna specifikacija se parsira u apstraktno sintaksno stablo kome je pridružena jedna od naslednica klase `Encoding` (bazna klasa i naslednice čine hijerarhiju klasa). Ova hijerarhija je prikazana na slici 2.13. Obilazak stabla se vrši u okviru faze *`meSAT`* `enkoder` (već spomenute u diskusiji slike 2.12) i tada se pozivaju metode pridružene klase naslednice. Tri klase



Slika 2.13: Hijerarhija klasa za kodiranje sistema *meSAT*.

koje se odnose na kodiranja na SAT nasleđuju baznu klasu, a još tri nasleđuju ove klase. Klasa `EncoderSmt` implementira svođenje na SMT, a klasa `EncoderFictive` služi za prikupljanje atributa instanci (više o ovome u glavi 3). Tokom obilaska se obavlja ono što je predviđeno klasom, dakle ili se generiše SAT/SMT instanca, ili se prikupljaju atributi ulazne CSP istance. Način za prevođenje se za svako ograničenje može definisati bilo za sva kodiranja, za grupacije sličnih kodiranja, ili za svako kodiranje pojedinačno.

Svođenje problema CSP na SMT je vrlo jednostavno implementirati, a prevođenje se vrši u teoriju linearne celobrojne aritmetike. Globalna ograničenja se razbijaju na jednostavnija ograničenja i tek onda prosleđuju SMT rešavaču. Efikasniji načini za korišćenje globalnih ograničenja sa SMT rešavačima postoje ([10]), ali ih ovde ne razmatramo.

Bulovska ograničenja kardinalnosti mogu se prevesti na SAT korišćenjem sekvencijalnih brojača [115] i mreža kardinalnosti [7]. Za prevođenje za *tačno-jedan-tačan* ograničenja, alternativno se mogu koristiti kodiranje proizvoda [32] i komandant kodiranje [76].

2.5.2 Eksperimentalna evaluacija

Da bismo pokazali prednosti mogućnosti korišćenja većeg broja kodiranja i testirali hipotezu da hibridna kodiranja unapređuju osnovna kodiranja, sproveli smo eksperimentalnu evaluaciju ([122]). Koristili smo sva kodiranja dostupna u okviru sistema *meSAT*, sekvencijalne unarne brojače za bulovska ograničenja kardinalnosti i kodiranje proizvoda za *tačno-jedan-tačan* ograničenja.

Instance. Koristili smo tri korpusa problema CSP: (i) CPAI09 korpus koji sadrži sve instance koje koriste globalna ograničenja sa takmičenja "Fourth International CSP Solver Competition"[19], (ii) MiniZinc korpus koji sadrži 29 problema iz MiniZinc distribucije[20], koje smo takođe kodirali tako da koriste globalna ograničenja, i (iii) instance problema *dominirajućih kraljica*, koji je opisan u katalogu globalnih ograničenja[21]. Globalna ograničenja korišćena u instancama su *all-different*, *count*, *cumulative*, *disjunctive*, *element*, *global_ cardinality*, *global_ cardinality_ with_ costs*, *lex_ less*, *lex_ less_ eq*, *nvalue* i *weighted_ sum*. Razlog za uključivanje samo instanci koje koriste globalna ograničenja jesu rezultati preliminarnih eksperimenata – oni su pokazali da je kodiranje uređenja najefikasnije kodiranje za instance korišćenih korpusa koje ne koriste globalna ograničenja: propagiranje je brže i generisane instance su manje. Rezultati eksperimenata predstavljenih u poglavlju 3.5.1 takođe su potvrdili ovu tvrdnju. Za svaki problem, navedeni korpusi sadrže nekoliko instanci koje se razlikuju u veličini problema i specifičnim ulaznim podacima. Koristili smo dva formata ulaznih datoteka: MiniZinc jezik [98] i Sugar jezik [124].

Postupak za prevođenje instanci između dva korišćena jezika i jezika XCSP detaljno je opisan u poglavlju 2.1.6. Instance CPAI09 korpusa su automatski prevedene iz originalnog ulaznog jezika u MiniZinc i u Sugar jezik. Deset instanci problema *nengfa* nisu mogle da budu prevedene pomoću programa `xcsp2mzn` i zbog toga su izostavljene iz eksperimenata. Instance MiniZinc korpusa su već u MiniZinc ulaznom formatu i koriste originalne ulazne podatke (navedene u .dzn datotekama iz MiniZinc distribucije). Nove specifikacije problema u Sugar jeziku su napravljene (.mzn datoteke nisu direktno korišćene zbog različitih vrsta ograničenja koja podržavaju ovi formati). Korpus Dominirajuće kraljice sadrži samo instance problema dominirajućih kraljica. Velika pažnja u sistemu *meSAT* je posvećeno efikasnom kodiranju globalnih ograničenja. Na primer, ograničenje *nvalue* može se vrlo efikasno kodirati kada se koriste direktno kodiranje ili kodiranje podrške. Pošto nijedan problem iz prva dva korpusa ne koristi ovo ograničenje, da bismo testirali efikasnost sistema *meSAT* na ovom vrstu ograničenja, uključili smo instance problema dominirajućih kraljica. Specifikacija ovog problema ista je i za MiniZinc i za Sugar jezike.

Interesantne instance. U našim preliminarnim eksperimentima smo primetili da su mnoge instance lake za većinu rešavača, a da postoje instance koje su previše

[19] `http://www.cril.univ-artois.fr/CPAI09`
[20] `http://www.minizinc.org`
[21] `http://sofdem.github.io/gccat/`

teške za sve rešavače. Zato se u većini eksperimenata fokusiramo na "interesantne" instance, one koje su rešene za vremensko ograničenje od 600 sekundi[22] od strane barem jedne, ali ne sve tri metode svođenja na SAT: pomoću kodiranja direktno-podrška, kodiranja uređenja i kodiranja direktno-uređenje implementiranih u okviru sistema *meSAT*. Ove instance su izdvojene da bi se stekla jasnija slika poređenja raznih kodiranja – izbegnute su preteške i prelake instance. Da bismo povećali broj interesantnih instanci, generisali smo instance većih dimenzija za probleme drugog korpusa kod kojih je većina instanci bila prelaka. Da bismo distribuciju interesantnih instanci među problemima učinili nešto pravilnijom nego u originalnom korpusu, za probleme za koje je postojalo više od 30 interesantnih instanci, na slučajan način smo odabrali 30 instanci koje su korišćene u eksperimentima.

Eksperimentalno okruženje. Svi testovi su vršeni na višeprocesorskom računaru sa 4 procesora AMD Opteron(tm) CPU 6168 na 1.9Ghz (svaki sa po 12 jezgara), sa 2GB RAM memorije po jezgru, pod operativnim sistemom Ubuntu server 2.6.32-38. U svim eksperimentima i kod svih sistema u ovom poglavlju, rešavač MiniSat 2.2 [45] je korišćen za rešavanje generisanih DIMACS instanci. Svakoj metodi rešavanja dato je na raspolaganje 600 sekundi za svaku instancu (uključeno je i vreme za kodiranje u ovo vreme). U svim tabelama simbol # označava ukupan broj instanci. U poljima tabela dat je broj rešenih instanci a vreme potrebno za rešavanje dato je u zagradi. U većini tabela nisu dati odvojeni rezultati za različite korpuse, već samo ukupni rezultati. Ukupno vreme rešavanja (u minutima) je prikazano u zagradama (za nerešenu instancu 600 sekundi je dodato na ukupno vreme). Prosečno vreme po instanci se može direktno izračunati kao količnik ukupnog vremena i broja instanci. Pošto je u nekim slučajevima rešeno manje od 50% instanci, medijana vremena nije prikazana. U svakom redu, sadržaj polja koje odgovara najboljem rešavaču je podebljan.

Tabela 2.2 prikazuje rezultate eksperimenata za sve instance dok tabela 2.3 prikazuje rezultate za interesantne instance. Tumačenje ovih rezultata sledi u naredna dva poglavlja.

[22] Ovo vremensko ograničenje je jedno od standardnih ograničenja koja se koriste u takmičenjima i radovima iz ove oblasti.

Tabela 2.2: Rezultati eksperimenata sistema *meSAT* (ds – kodiranje direktno-podrška, o – kodiranje uređenja, do – kodiranje direktno-uređenje), Sugar, Azucar (m2 – kompaktno kodiranje uređenja, log – logaritamsko kodiranje), `mzn-g12cpx` i `mzn-g12lazy`.

		meSAT			Sugar	Azucar		`mzn-g12`	
korpus	#	ds	o	do	Sugar	m2	log	cpx	lazy
CPAI09	583	359 (2402)	488(1099)	486 (1104)	**494(1053)**	448(1503)	428(1722)	332(2609)	348(2638)
MiniZinc	770	**583(2100)**	562(2516)	579 (2266)	554 (2572)	509(2965)	486(3309)	371(4170)	492(2974)
D. kraljice	26	**23 (76)**	2 (240)	20 (89)	3 (236)	2 (246)	2 (247)	4 (224)	5 (211)
Ukupno	1379	965 (4578)	1052(3855)	**1085(3459)**	1051 (3862)	959(4713)	916(5278)	707(7002)	845(5824)

Tabela 2.3: Rezultati eksperimenata za interesantne instance podeljene po kategorijama problema. Za svaki problem početno slovo označava korpus (C za CPAI09, M za MiniZinc, D za Dominirajuće kraljice). Korišćen je isti skup rešavača kao u tabeli 2.2.

		meSAT			Sugar	Azucar		`mzn-g12`	
problem	#	ds	o	do	Sugar	m2	log	`cpx`	`lazy`
C/allsquares	16	0 (160)	16 (48)	15 (47)	**16 (7)**	11 (71)	9 (93)	0 (160)	0 (160)
C/bibd	16	0 (160)	**16 (23)**	**16 (23)**	15 (16)	15 (25)	13 (55)	3 (149)	1 (151)
C/CabinetStart1	30	0 (300)	30 (16)	30 (14)	30 (16)	30 (9)	**30 (3)**	30 (5)	30 (17)
C/compet08	2	**2 (0)**	0 (20)	**2 (0)**	1 (11)	0 (20)	0 (20)	0 (20)	0 (20)
C/costasArray	2	**2 (10)**	1 (11)	0 (20)	1 (16)	0 (20)	0 (20)	0 (20)	0 (20)
C/latinSquare	1	0 (10)	**1 (0)**	0 (10)	**1 (0)**	0 (10)	0 (10)	0 (10)	0 (10)
C/magicSquare	6	0 (60)	**6 (11)**	4 (25)	5 (25)	4 (23)	3 (38)	0 (60)	0 (60)
C/ortholatin	1	0 (10)	0 (10)	1 (8)	1 (4)	**1 (1)**	1 (8)	0 (10)	0 (10)
C/pseudoGLB	30	0 (300)	**30 (8)**	**30 (8)**	30 (15)	29 (30)	24 (93)	28 (35)	26 (125)
C/QG	1	0 (10)	1 (7)	1 (6)	**1 (2)**	**1 (2)**	0 (10)	0 (10)	0 (10)
M/bacp	5	0 (50)	5 (13)	5 (8)	5 (14)	4 (26)	3 (34)	**5 (1)**	**5 (1)**
M/bibd	2	2 (10)	0 (20)	2 (9)	**2 (1)**	1 (10)	1 (10)	1 (11)	1 (11)
M/cars	9	**9 (21)**	0 (90)	5 (74)	4 (80)	0 (90)	1 (84)	0 (90)	0 (90)
M/carseq	8	8 (25)	0 (80)	5 (69)	3 (71)	0 (80)	1 (79)	0 (80)	**8 (1)**
M/costas-array	4	4 (3)	0 (40)	4 (26)	0 (40)	**4 (1)**	0 (40)	0 (40)	0 (40)
M/debruijnbinary	2	0 (20)	2 (1)	**2 (0)**	2 (1)	2 (1)	**2 (0)**	**2 (0)**	**2 (0)**
M/golfers	2	1 (10)	2 (5)	0 (20)	2 (2)	2 (10)	1 (16)	1 (10)	**2 (1)**
M/golomb	1	0 (10)	1 (7)	**1 (4)**	1 (9)	1 (9)	0 (10)	**1 (4)**	**1 (4)**
M/knights	14	**14 (41)**	0(140)	4 (126)	0(140)	0 (140)	0 (140)	12 (20)	5 (91)
M/langford	2	0 (20)	2 (2)	**2 (0)**	**2 (0)**	2 (3)	2 (15)	2 (3)	2 (3)
M/latinsquares	10	0 (100)	**10 (5)**	0 (100)	9 (22)	0 (100)	0 (100)	4 (66)	5 (67)
M/magicseq	2	0 (20)	2 (3)	2 (3)	**2 (1)**	**2 (1)**	2 (2)	**2 (1)**	**2 (1)**
M/nsp	2	**2 (0)**	0 (20)	**2 (0)**	1 (15)	2 (4)	2 (7)	0 (20)	0 (20)
M/QCP	1	**1 (1)**	0 (10)	**1 (1)**	1 (5)	0 (10)	0 (10)	0 (10)	0 (10)
M/queens	9	8 (39)	0 (90)	**9 (32)**	2 (74)	3 (70)	3 (72)	0 (90)	0 (90)
M/searchstress	2	0 (20)	2 (14)	0 (20)	2 (4)	0 (20)	0 (20)	**2 (0)**	**2 (0)**
M/steiner	2	**2 (0)**	1 (11)	0 (20)	1 (10)	1 (11)	1 (14)	**2 (0)**	1 (10)
M/stilllife	2	0 (20)	2 (9)	**2 (8)**	0 (20)	0 (20)	0 (20)	0 (20)	0 (20)
M/talentscheduling	1	0 (10)	**1 (1)**	1 (2)	1 (4)	1 (10)	0 (10)	1 (6)	0 (10)
D/dominatingQueens	23	**21 (66)**	0(230)	18 (79)	1(225)	0 (230)	0 (230)	2 (213)	3 (201)
Ukupno	208	76(1507)	131(945)	**164(763)**	142(849)	116(1057)	99(1263)	98(1165)	96(1255)

2.5.3 Poređenje sistema *meSAT* sa ostalim savremenim sistemima

U ovom poglavlju poredimo rezultate sistema *meSAT* sa rezultatima nekih savremenih sistema. U poređenju su korišćeni sistemi Sugar-v2-0-0, Azucar-v0.2.3 [128] (korišćen u dve različite konfiguracije – Azucar-m2 koji koristi kompaktno kodiranje uređenja za $m = 2$ i koja je podrazumevana konfiguracija ovog sistema, i Azucar-log koji implementira logaritamsko kodiranje), kao i dva sistema koji koriste lenjo generisanje klauza i uključeni su u MiniZinc 1.6 distribuciju: `mzn-g12cpx` i `mzn-g12lazy`.

Rezultati dati u tabeli 2.2 i tabeli 2.3 pokazuju da je na korišćenim korpusima *meSAT* značajno efikasniji od ostalih sistema. Izuzetak je Sugar (sistem koji je pobeđivao na više CSP takmičenja) – *meSAT* je bio efikasniji od ovog sistema jedino u slučaju korišćenja kodiranja direktno-uređenje. Pošto razlika nije velika, ona se može pripisati i slučajnim varijacijama u SAT rešavanju[23].

2.5.4 Poređenje različitih kodiranja u okviru sistema *meSAT*

U ovom poglavlju je izvršeno poređenje efikasnosti kodiranja implementiranih u okviru sistema *meSAT*: direktnog kodiranja (d), kodiranja podrške (s), kodiranja direktno-podrška (ds), kodiranja uređenja (o) i kodiranja direktno-uređenje (do)[24]. Cilj je poređenje hibridnih kodiranja direktno-podrška i direktno-uređenje sa kodiranjima nad kojima su izgrađena (direktno kodiranje i kodiranje podrške, odnosno direktno kodiranje i kodiranje uređenja) na interesantnim instancama. Rezultati poređenja kodiranja su dati u tabeli 2.4.

Rezultati pokazuju da je kodiranje direktno-uređenje neznatno efikasnije od direktnog kodiranja i kodiranja uređenja na korišćenim instancama, ali ovo može biti pripisano i slučajnim varijacijama u vremenima SAT rešavanja. Pošto je razlika mala, posebni rezultati za direktno kodiranje i kodiranje podrške neće nadalje biti razmatrani. Rezultati takođe pokazuju da je kodiranje direktno-uređenje efikasnije

[23] Vreme SAT rešavanja uvek ima određenu dozu nepredvidivosti i na njega se može uticati trivijalnim promenama na instancama koje se rešavaju (na primer, promenom redosleda klauza), pa postoji mala doza slučajnosti u svim eksperimentalnim rezultatima koji uključuju SAT rešavanje. Statistički pristup poređenja SAT rešavača je razvio Nikolić [100].

[24] Skraćenice potiču od engleskih naziva ovih kodiranja.

Tabela 2.4: Ukupni rezultati poređenja različitih kodiranja u okviru sistema *me-SAT* na interesantnim instancama.

korpus	#	d	s	ds	o	do
Ukupno	208	61 (1608)	71 (1528)	76 (1507)	131 (945)	**164 (763)**

od kodiranja uređenja i kodiranja direktno-podrška na korišćenim instancama. Ovi rezultati potvrđuju potrebu za razvojem hibridnih kodiranja.

Ukupni rezultati dati u tabeli 2.4 mogu sugerisati da su kodiranja zasnovana na kodiranju uređenja (kodiranje direktno-uređenje može se smatrati zasnovanim na kodiranju uređenja pošto je u mnogim slučajevima jednako kodiranju uređenja) značajno efikasnija od kodiranja direktno-podrška. Međutim, rezultati po problemima dati u tabeli 2.3 pokazuju da postoje mnogi problemi koji su pogodniji za kodiranje direktno-podrška (na primer, M/cars, M/carseq, M/knights, D/dominatingQueens). Tabela 2.5 prikazuje za svako kodiranje broj interesantnih instanci na kojima je to kodiranje bilo najbolje i pokazuje da za svako kodiranje postoji veliki broj instanci na kojima je ono najbolje. Ovi rezultati pokazuju da su različita kodiranja pogodno za različite probleme.

Tabela 2.5: Broj interesantnih instanci na kojima je svako od kodiranja postiglo najbolji rezultat.

#	ds	o	do
208	59	64	**85**

Rezultati u tabeli 2.3 takođe pokazuju da kodiranje direktno-uređenje ni na jednom problemu nije izrazito dominantno, kao što je to slučaj sa kodiranjem direktno-podrška (M/cars, M/knights) i kodiranjem uređenja (M/latinsquares). Razlog je sama konstrukcija ovog kodiranja: teži se izbegavanju lošeg kodiranja po cenu da se dobiju i nešto lošiji rezultati nego da je korišćeno samo jedno od kodiranja u njegovom sastavu.

2.6 Analiza kodiranja

U ovom poglavlju ćemo razmatrati odnos različitih kodiranja i osnovnih tehnika rada SAT rešavača. Potom ćemo navesti primere kodiranja na konkretnim problemima i dati neke smernice za kodiranje problema u praksi.

2.6.1 Analiza kodiranja na osnovu rada SAT rešavača

U ovom poglavlju ćemo izvršiti analizu rada nekih mehanizama SAT rešavača (oni su opisani u poglavljima 2.3.1.2 i 2.3.1.3) prilikom obrade formula dobijenih kodiranjima. Cilj je utvrditi povezanost kodiranja promenljivih i mehanizama SAT rešavanja, da bi se videlo u kojim slučajevima i zašto su neka kodiranja pogodnija od drugih za predstavljanje promenljivih. Konkretno, razmotrićemo šta se dešava kod različitih kodiranja kada jedna od iskaznih promenljivih kojima se predstavlja celobrojna promenljiva dobije vrednost. Dva pravila koja su ključna i koja se najviše primenjuju u ovoj situaciji su `Decide` i `Unit Propagate`. Opis koji sledi se stoga samo odnosi na kodiranje promenljivih, a ne i na kodiranje ograničenja. Kada je u pitanju kodiranje ograničenja, stvar se značajno komplikuje i detaljno istraživanje o tome šta se dešava u raznim slučajevima bi mogao da bude potpuno nov pravac istraživanja.

Direktno kodiranje i kodiranje podrške. Pretpostavimo da je za predstavljanje promenljive x_i iskorišćeno naivno prevođenje bulovskog ograničenja kardinalnosti u iskaznu formulu. Dakle, promenljiva x_i je predstavljena iskaznim promenljivama $p_{i,l}, \ldots, p_{i,u}$ a ograničenje $p_{i,l} + \ldots + p_{i,u} = 1$ je prevedeno u klauzu $c = p_{i,l} \vee \ldots \vee p_{i,u}$ (barem jedna iskazna promenljiva je tačna) i skup klauza $S = \{\neg p_{i,v} \vee \neg p_{i,w}, 1 \leq v < w \leq n\}$ (ne mogu dve iskazne promenljive biti istovremeno tačne).

Kada neka od promenljivih $p_{i,v}$ ($v \in \{l, \ldots, u\}$) dobije vrednost 1, onda klauza c postaje zadovoljena, dok sve klauze iz skupa S koje sadrže promenljivu $p_{i,v}$ ($\neg p_{i,l} \vee \neg p_{i,v}$, $\ldots$, $\neg p_{i,v-1} \vee \neg p_{i,v}$, $\neg p_{i,v+1} \vee \neg p_{i,v}$, $\ldots$, $\neg p_{i,u} \vee \neg p_{i,v}$) postaju takve da im je tačno po jedan (drugi) od dva literala netačan ($\neg p_{i,v}$). Propagiranjem jediničnih klauza se prvi literal u tim klauzam postavlja da ima vrednost 1, što znači da sve promenljive $p_{i,l}, \ldots, p_{i,v-1}, p_{i,v+1}, \ldots, p_{i,u}$ dobijaju vrednost 0. Ovime se postiže da sve klauze iz S koje ne sadrže promenljivu $p_{i,v}$ postaju zadovoljene. Dakle, kada jedna od iskaznih promenljivih $p_{i,v}$ koje se odnose na promenljivu x_i dobije vrednost 1, sve ostale promenljive korišćene za predstavljanje te celobrojne promenljive dobiju vrednost 0, a sve klauze koje se odnose na predstavljanje promenljive x_i postaju zadovoljene.

Kada neka od promenljivih $p_{i,v}$ dobije vrednost 0, onda u klauzi koja sadrži literal $p_{i,v}$, taj literal postaje netačan, dok klauze koje sadrže literal $\neg p_{i,v}$ postaju zadovoljene.

Kodiranje uređenja. Smatramo da je promenljiva x_i kodirana pomoću promenljivih $p_{i,l-1}, \ldots, p_{i,u}$, i da su uvedene klauze $p_{i,v-1} \rightarrow p_{i,v}$ za svako $v \in \{l_i, \ldots, u_i\}$.

Kada neka od promenljivih $p_{i,v}$ dobije vrednost 1 (što označava da je $x_i \leq v$), onda klauza $p_{i,v} \rightarrow p_{i,v+1}$ utiče na to da i promenljiva $p_{i,v+1}$ dobije vrednost 1. Na isti način redom promenljive $p_{i,v+2}, \ldots, p_{i,u-1}$ dobijaju vrednost 1 (promenljiva $p_{i,u}$ je u startu imala dodeljenu vrednost 1).

Kada neka od promenljivih $p_{i,v}$ dobije vrednost 0 (što označava da je $x_i > v$), onda klauza $p_{i,v-1} \rightarrow p_{i,v}$ utiče na to da i promenljiva $p_{i,v-1}$ dobije vrednost 0. Na isti način redom promenljive $p_{i,v-2}, \ldots, p_{i,l}$ dobijaju vrednost 0 (promenljiva $p_{i,l-1}$ je u startu imala dodeljenu vrednost 0).

Logaritamsko kodiranje. Već je rečeno da ovo kodiranje odgovara predstavljanju brojeva u računaru i da je za neku celobrojnu promenljivu x_i iskazna promenljiva $p_{i,k}$ tačna akko je k-ti bit binarne reprezentacije vrednosti dodeljene x_i jednak 1. Kod ovog kodiranja se pomoću propagiranja jediničnih klauza u opštem slučaju na osnovu dodeljene vrednosti nekoj promenljivoj $p_{i,k}$ ne mogu dobiti vrednosti ostalih promenljivih, tj. bilo da $p_{i,k}$ dobije vrednost 0 ili 1, ne može se ništa zaključiti o vrednostima ostalih promenljivih.

Poređenje kodiranja. Kod direktnog kodiranja i kodiranja podrške se postavljanjem na 1 vrednosti neke iz skupa iskaznih promenljivih koje predstavljaju celobrojnu promenljivu vrlo brzo dobijaju vrednosti i ostalih iskaznih promenljivih tog skupa a sve odgovarajuće klauze postaju zadovoljene. Postavljanjem vrednosti na 0 se postiže mnogo slabiji efekat i ova vrednost ne utiče neposredno na vrednosti drugih iskaznih promenljivih.

Kod kodiranja uređenja se postavljanjem na vrednost 1 ili 0 neke iz skupa iskaznih promenljivih koje predstavljaju celobrojnu promenljivu u proseku postiže isti efekat: polovina iz tog skupa iskaznih promenljivih dobija tu vrednost.

Kod logaritamskog kodiranja se postavljanjem na 1 ili 0 vrednosti neke iz skupa iskaznih promenljivih koje predstavljaju celobrojnu promenljivu ne utiče neposredno na vrednosti drugih promenljivih iz tog skupa.

Na osnovu navedenog se može zaključiti da posmatrajući samo kodiranje celobrojnih promenljivih i direktno kodiranje (kodiranje podrške) i kodiranje uređenja imaju prednosti nad ostalim kodiranjima u određenim situacijama, dok je logaritamsko kodiranje nepogodnije za osnovne savremene mehanizme rada SAT rešavača.

Naglasimo da su diskutovani samo elementarni slučajevi i da se u praksi diskusija može značajno zakomplikovati. Na primer, kod direktnog kodiranja se po pravilu koriste drugi (pogodniji) načini za prevođenje bulovskih ograničenja kardinalnosti (neka su pomenuta u poglavlju 2.3.2) kod kojih se uvode nove promenljive. U zavisnosti od načina uvođenja ovih promenljivih zavisi i šta se dešava kada se nekoj od njih dodeli vrednost 1 ili 0. Treba naglasiti da i u slučaju kada veličine domena promenljivih pređu određenu granicu, logaritamsko kodiranje zbog svoje kompaktnosti obično postaje jedina opcija za kodiranje celobrojnih promenljivih, pa postoje situacije kada je ono pogodnije od ostalih kodiranja.

2.6.2 Diskusija kodiranja na konkretnim problemima

U ovom poglavlju ćemo pokušati da damo objašnjenje efikasnosti kodiranja na pojedinačnim problemima, sa ciljem da u narednom poglavlju damo neke smernice za korišćenje različitih kodiranja. Objašnjenje se zasniva na rezultatima predstavljenim u tabeli 2.3.

2.6.2.1 Problem određivanja redosleda proizvodnje automobila (M/cars)

Ovaj problem se obično naziva *problem određivanja redosleda proizvodnje automobila*, jedan je od najčešće rešavanih problema CSP i prvi problem u biblioteci ovih problema, CSPLIB [57]. Potrebno je proizvesti određen broj automobila, pri čemu ti automobili predstavljaju različite varijacije osnovnog modela, u zavisnosti od opreme koju poseduju (npr. dodatna oprema se može odnositi na posedovanje klime, bočnih vazdušnih jastuka itd.). Različita odeljenja ugrađuju različitu opremu. Svako od odeljenja ima određeni kapacitet, tj. može opslužiti određeni procenat automobila u nekom periodu.

Primer 15 (Primer problema određivanja redosleda proizvodnje automobila)**.** *Jedan primer ulaznih parametara ovog problema je:*

```
n = 10, m = 5, v = 6
(1,2), (2,3), (1,3), (2,5), (1,5)
1. [1]  1 0 1 1 0
2. [1]  0 0 0 1 0
3. [2]  0 1 0 0 1
```

```
4. [2]  0 1 0 1 0
5. [2]  1 0 1 0 0
6. [2]  1 1 0 0 0
```

U prvoj liniji dat je broj automobila (10), broj mogućnosti za dodatnu opremu (5) i broj različitih varijacija baznog modela (6) koji se proizvodi. U drugoj vrsti dat je niz uređenih parova jednak broju odeljenja pri čemu par (p, q) *(npr. (2,3)) označava da u q (3) uzastopnih vremenskih jedinica, odgovarajuće odeljenje r (2) može obrađivati maksimalno p (2) automobila. Naredne vrste za svaku varijaciju automobila (dat je indeks na početku vrste) sadrže broj tih automobila koje je potrebno proizvesti (npr. potrebno je proizvesti 2 automobila sa indeksom 3), i potom za svaku od mogućnosti za dodatnu opremu, 1 ako ta varijacija sadrži tu opremu, a 0 ako ne sadrži.*

Jedno rešenje problema je sledeći redosled proizvodnje automobila, pri čemu je prikazana i matrica sa elementima $b_{i,j}$ *(*$i \in [1, 10], j \in [1, 5]$*) koji odgovaraju dodatnoj opremi automobila:*

```
1.   1 0 1 1 0
2.   0 0 0 1 0
6.   1 1 0 0 0
3.   0 1 0 0 1
5.   1 0 1 0 0
4.   0 1 0 1 0
4.   0 1 0 1 0
5.   1 0 1 0 0
3.   0 1 0 0 1
6.   1 1 0 0 0
```

Matrica b nam omogućava da proverimo da odeljenje ne prevazilaze svoj kapacitet. Na primer, za opciju 5 vidimo da se ni u kojih 5 uzastopnih vremenskih jedinica ne javlja više od jednog automobila.

Jedan CSP model ovog problema uvodi promenljive $a_1, \ldots, a_n$ sa domenom $[1, v]$ koje označavaju redosled proizvodnje automobila. Uvode se i promenljive $b_{i,j}$ ($i \in [1, n], j \in [1, m]$) sa domenom $[0, 1]$ koje imaju isto značenje kao i u prethodnom primeru:

$$\begin{matrix} b_{1,1} & \dots & b_{1,m} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \dots & b_{n,m} \end{matrix}$$

Potrebno je uvesti implikacije koje povezuju nizove promenljivih a i b: ako je u nizu a na nekoj poziciji odgovarajuća varijacija, onda u istoj vrsti matrice b mora stajati odgovarajući niz nula i jedinica koji odgovara toj varijaciji.

Za svaki uređeni par (p, q) koji odgovara koloni r i koji je dat specifikacijom problema, uvodi se na odgovarajućoj koloni niz ograničenja:

$$\begin{gathered} count(\{b_{1,r}, \dots, b_{q,r}\},\ 1) = p \\ count(\{b_{2,r}, \dots, b_{q+1,r}\},\ 1) = p \\ \vdots \\ count(\{b_{n-q+1,r}, \dots, b_{n,r}\},\ 1) = p \end{gathered}$$

Takođe, uvode se i ograničenja koja označavaju koliko je kojih automobila potrebno proizvesti. Npr. ako je potrebno proizvesti f automobila sa indeksom g, onda se uvodi ograničenje $count(\{a_1, \dots, a_n\},\ g) = f$.

Razmotrimo promenljive $b_{i,j}$ i *count* ograničenja na njima. Ove promenljive su iz domena sa malom kardinalnošću (vrednosti su iz skupa $[0, 1]$) pa na ovom problemu nije presudan odabir kodiranja na promenljivama, već na ograničenjima. Primetimo da se svako od ograničenja *count* može zameniti odgovarajućim uslovom koji uključuje samo sabiranje (npr., $b_{1,j} + \ldots + b_{q,j}\} = p$. Upravo od odluke da li koristiti ograničenje *count* ili aritmetiku zavisi i koje će kodiranje biti uspešnije na ovom problemu. U slučaju ograničenja *count*, kodiranje direktno-podrška ima prednost u odnosu na kodiranje uređenja (opisano u poglavlju 2.4.2.1), dok je kodiranje direktno-uređenje lošije od kodiranje direktno-uređenje jer se vreme i prostor dodatno upotrebljavaju za dvostruko kodiranje celobrojnih promenljivih kao i za klauze povezivanja. U slučaju predstavljanja ograničenja pomoću sabiranja, bolje je koristiti kodiranje uređenja, jer je ono najpogodnije za aritmetičke operacije.

Naredna 2 problema ćemo opisati samo u vrlo kratkim crtama.

2.6.2.2 Problem dominirajućih kraljica (D/dominatingQueens)

Kraljica dominira nad nekim poljem na šahovskoj tabli ako se to polje nalazi u istoj vrsti, koloni ili dijagonali na kojoj i kraljica (kraljica dominira nad nekim poljem ako se na praznoj tabli u jednom šahovskom potezu može pomeriti na to polje, ili

ako se već nalazi na tom polju). Problem je postaviti m kraljica na šahovsku tablu dimenzija $n \times n$, tako da nad svakim poljem dominira barem jedna kraljica (otuda i ime problema). Na primer, ako se na šahovsku tablu dimenzija 3×3 postavi kraljica na središnje polje, onda ta kraljica dominira nad svim poljima te table. Postoje dve varijacije ovog problema. U prvoj, postavljene kraljice se ne smeju međusobno napadati. Ako je vrednost broja m jednaka dimenziji table n, onda ova varijacija problema jeste jednaka problemu n kraljica opisanom u poglavlju 2.1.3. U drugoj varijaciji, vrednost broja m nije data, već je potrebno odrediti minimalan broj kraljica koje mogu dominirati nad tablom dimenzija $n \times n$. Dakle, u toj varijaciji radi se o problemu COP. Problem dominirajućih kraljica u bilo kojoj od navedenih varijanti jeste specijalan slučaj *problema pokrivanja skupa* (eng. *set covering problem*), koji je dosta izučavan [30].

Specifikacija problema sastoji se iz navođenja domena celobrojnih promenljivih i zadavanja ograničenja *nvalue* nad ovim promenljivama. Kodiranje uređenja je neefikasno u rešavanju instanci ovog problema pošto ono ne može na specijalan način da kodira ograničenje *nvalue*. Najefikasnije je kodiranje direktno-podrška, pošto ono na ovom problemu intenzivno koristi specijalizovano ograničenje *nvalue*. Hibridno kodiranje koje se sastoji iz prethodna dva kodiranja (kodiranje direktno-uređenje) daje nešto lošije rezultate nego kodiranje direktno-podrška. Razlog verovatno leži u dvostrukom kodiranju svih korišćenih promenljivih, što uvek dovodi do većih instanci i većeg vremena potrebnog za rešavanje ovih instanci. Kod ovog problema celobrojne promenljive imaju velike domene, pa su formule dobijene dvostrukim kodiranjem značajno veće.

2.6.2.3 Problem bibd (C/bibd)

Specifikacija ovog problema sastoji se iz navođenja domena celobrojnih promenljivih i korišćenja dve vrste ograničenja: jednakosti izraza koji predstavljaju aritmetičke operacije nad promenljivama i od globalnih ograničenja koja su takođe zasnovana na aritmetičkim operacijama i za koje, po našem saznanju, ne postoje specijalno direktno kodiranje. Pošto je kodiranje uređenja efikasnije u obradi aritmetike, ono i daje najbolje rezultate na ovom problemu.

2.6.3 Neke smernice za kodiranje problema

Kodiranja su intenzivno proučavana prethodnih godina i davanje nekih konkretnih pravila za to koje od njih koristiti u određenim slučajevima je vrlo težak i nezahvalan zadatak. Zadatak postaje još teži kada se u problemu koji se rešava javlja veliki broj raznorodnih ograničenja. Na osnovu prethodnih delova, možemo doći do određenih smernica o pogodnim kodiranjima određenih ograničenja:

- U slučaju aritmetičkih ograničenja, efikasnije je koristiti kodiranje uređenja u odnosu na direktno kodiranje i kodiranje podrške.
- U slučaju specijalizovanih globalnih ograničenja, pogodnije je koristiti direktno kodiranje/kodiranje podrške nego kodiranje uređenja. Razlog ovome je veliki broj efikasnih kodiranja bulovskih ograničenja kardinalnosti, koja se intenzivno koriste kod ovih specijalizovanih ograničenja.
- U slučaju većeg domena celobrojnih promenljivih, pogodnije je koristiti kodiranje uređenja, jer su klauze generisane ovim kodiranjem uvek iste veličine i ima ih znatno manje.

Ako se problem sastoji iz kombinacije ograničenja, kodiranje direktno-uređenje može predstavljati najbolju opciju. Izuzetak od ovoga može biti situacija kada promenljive imaju veliki domen, jer se u tom slučaju može desiti da najveći deo generisane instance predstavlja dvostruko kodiranje istih celobrojnih promenljivih.

Glava 3

Portfolio pristup za rešavanje problema CSP

U oblasti veštačke inteligencije a i u širem kontekstu računarstva je opštepoznato da algoritmi imaju različite performanse pri rešavanju različitih problema i da ne postoji algoritam koji je pogodan za sve tipove problema. Za neki problem i skup algoritama za njegovo rešavanje, *problem izbora algoritma* (eng. *algorithm selection problem*) [107, 20] se sastoji u odabiru algoritma koji će biti najefikasniji u rešavanju tog problema. Postupak merenja efikasnosti algoritama nije jednoznačan i potrebno ga je utvrditi.

Poznato je da isti algoritam može imati vrlo različite performanse u rešavanju nekog problema, u zavisnosti od vrednosti parametara tog algoritma. *Problem konfigurisanja algoritma* (eng. *algorithm configuration problem*) [66] se sastoji u odabiru konfiguracije algoritma koja je najpogodnija za problem koji se rešava. Dakle, fiksiran je algoritam koji će biti korišćen ali je potrebno utvrditi neke (ili sve) parametre algoritma.

U ovoj glavi se fokusiramo na *algoritamske portfolije* (eng. *algorithm portfolios*). Portfolio na raspolaganju ima više algoritama i u zavisnosti od konkretnog problema koji treba rešiti, zadatak portfolija je odabir jednog ili više algoritama koji bi imao/imali dobre performanse na datom problemu. Pojam algoritamskog portfolija se lako može proširiti da obuhvata i problem konfigurisanja algoritma – dovoljno je svaku konfiguraciju algoritma smatrati posebnim algoritmom. Postoje i specijalizovani pristupi za rešavanje problema konfigurisanja algoritma nevezani za algoritamske portfolije, na primer, korišćenje lokalne pretrage za postizanje optimalnih parametara algoritma [64]. Postoji veliki broj radova na temu problema izbora i

problema konfiguracije algoritama. Razvijene su biblioteke ASlib (Algorithm Selection Library)[20] i AClib (Algorithm Configuration Library)[66] koje sadrže skupove instanci i nekoliko standardnih portfolija. Neki od problema za koje postoje instance i portfoliji u ovim bibliotekama su SAT, Answer Set Programming, Mixed Integer Programming, problemi planiranja, itd. Ove biblioteke kao glavni cilj imaju obezbeđivanje fer poređenja različitih portfolija.

U fokusu ove glave su portfoliji za odabir jednog ili više rešavača problema CSP.

3.1 Tehnike mašinskog učenja

Dobar odabir rešavača (jednog ili više iz nekog unapred određenog skupa) je fundamentalan razlog uspeha savremenih portfolija i obično se odabir vrši korišćenjem tehnika *mašinskog učenja* (eng. *machine learning*), skraćeno ML. Postoji više definicija mašinskog učenja. Po jednoj, to je disciplina koja se bavi izgradnjom prilagodljivih računarskih sistema koji su sposobni da poboljšavaju svoje performanse koristeći informacije iz iskustva. Po drugoj, to je disciplina koja se bavi proučavanjem generalizacije i konstrukcijom i analizom algoritama koji generalizuju.

3.1.1 Osnovni pojmovi

Regresija i klasifikacija. Jedan od glavnih problema u oblasti mašinskog učenja je problem *regresije* (eng. *regression*) – predviđanja vrednosti *kontinualne* ciljne promenljive na osnovu vrednosti nekih drugih srodnih promenljivih, koje zovemo *atributi* (eng. *features*) i koje mogu biti bilo diskretne ili kontinualne. Nadalje se pretpostavlja da govorimo o *nadgledanom učenju* (to je oblik učenja kod kojeg je uz atribute data i neka vrednost koja se kasnije predviđa, tj. vrednost ciljne promenljive). Za razliku od regresije, *klasifikacijom* (eng. *classification*) se na osnovu atributa predviđa vrednost ciljne promenljive koja uzima vrednosti iz nekog konačnog skupa klasa. Na primer, regresija se može koristiti da se predvidi vreme rešavanja nekog rešavača na nekom problemu, dok se klasifikacija može koristiti da se predvidi kojoj kategoriji će pripasti vreme rešavanja rešavača na nekom problemu (na primer, kratko, srednje ili dugo rešavanje). Postoje različite metode za rešavanje problema regresije i klasifikacije. Obično se određuje neka vrsta statističkog modela koji izražava zavisnost ciljne promenljive od atributa (model se može smatrati funkcijom koja slika n-torku vrednosti atributa u vrednosti ciljne promenljive).

Naglasimo da model u ovoj glavi ima drugo značenje u odnosu na model uveden kod problema CSP. Koeficijenti modela se određuju korišćenjem skupa instanci na kojima su poznate vrednosti i ciljne promenljive i atributa.

Metaparametri i faza treniranja. Pre nego što se algoritamski portfolio može upotrebiti (bilo za regresiju ili klasifikaciju), potrebno ga je *trenirati*. U fazi treniranja su na raspolaganju vrednosti ciljne promenljive za određene kombinacije vrednosti atributa. Ove vrednosti su obično dobijene na nekim dostpunim instancama problema koji se rešava i te instance zajedno sa vrednostima atributa i ciljne promenljive čine *trening skup*. Na osnovu ovih podataka, pomoću algoritama treniranja se formira model, koji treba da što preciznije odredi vrednost ciljne promenljive na osnovu vrednosti atributa (uzimajući u obzir samo podatke dostupne u fazi treniranja). Model ima jedan ili više mogućih *metaparametara* koji predstavljaju karakteristike tog modela. Pre korišćenja algoritma treniranja, korisnik za svaki metaparametar određuje skup mogućih vrednosti, a algoritam uz pomoć eksperimentalne evaluacije određuje vrednost za svaki od metaparametara. Određivanje vrednosti metaparametara je netrivijalan proces i najbolji izbor ovih vrednosti se obično postiže poređenjem modela koji se dobijaju za različite vrednosti metaparametara. Na primer, metoda *k-najbližih suseda* koristi k najbližih instanci za određivanje vrednosti ciljne promenljive (više o ovome u sledećem poglavlju). Jedan metaparametar te metode je k, tj. potrebno je odrediti vrednost k tako da se dobije što je moguće preciznije određivanje vrednosti ciljne promenljive na osnovu vrednosti atributa (na primer, može biti mnogo pogodnije izabrati vrednost $k = 5$ nego $k = 30$).

Evaluacija modela. U cilju provere da li modeli koje daju tehnike mašinskog učenja imaju dobru moć *generalizacije*, potrebno je testirati ih na skupu podataka različitom od trening skupa, tj. na nekom *test skupu*. Procena se obično izvršava na jedan od dva načina. Prvi podrazumeva deljenje dostupnih podataka na trening i test skup, izvršavanje treninga uz dobijanje modela na trening skupu i potom merenje uspešnosti procene ciljne promenljive na test skupu. Pitanje koje se postavlja je kako pravilno podeliti podatke na trening i test skup, jer različite podele mogu uticati na veoma različite procene uspešnosti. Da bi se ovo pitanje izbeglo, koristi se drugi način procenjivanja – *unakrsna provera* (eng. *cross validation*). Dostupni podaci se dele u k delova (eng. *folds*). Dobijeni model treniranjem na $k - 1$ ostalih

delova se koristi za predviđanje vrednosti ciljne promenljive na preostalom delu. Na kraju, kada su dostupna predviđanja za sve instance, nad tim predviđanjima i nad stvarnim vrednostima se računaju mere kvaliteta predviđanja. Unakrsna provera daje pouzdaniju procenu mogućnosti generalizacije modela, ali je svakako više vremenski zahtevna.

3.1.2 Regresione i klasifikacione metode

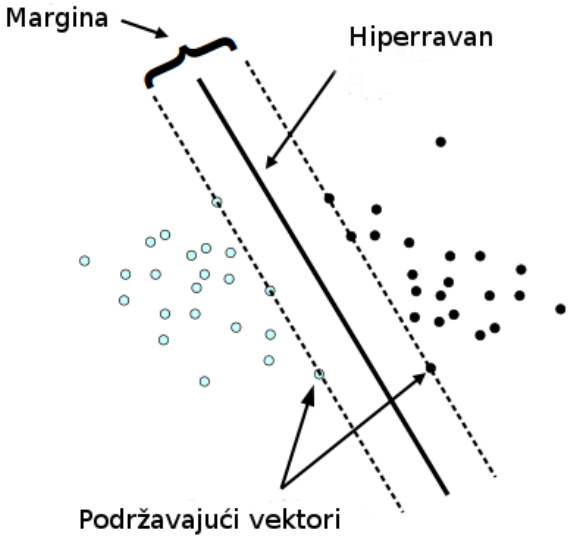
Opisaćemo neke poznate regresione i klasifikacione metode koje koriste atribute od kojih svaki može uzimati bilo diskretne ili kontinualne vrednosti.

Linearna regresija je jedna od najčešće korišćenih metoda mašinskog učenja. Ciljna promenljiva se modelira pomoću linearne kombinacije atributa [21]. Koeficijenti modela mogu se odrediti pomoću dobro poznate metode najmanjih kvadrata, koja minimizuje sumu kvadrata razlika između predviđenih i stvarnih vrednosti ciljne promenljive na trening skupu.

U metodi *k-najbližih suseda* (eng. *k-nearest neighbors*), skraćeno *k-NN*, se podrazumeva da je vrednost ciljne promenljive u nekoj tački jednaka linearnoj kombinaciji njenih vrednosti na k instanci trening skupa koje su najbliže ulaznoj instanci, u odnosu na neku funkciju udaljenosti definisanu nad vektorima atributa. Broj k se empirijski određuje. Za koeficijente linearne kombinacije najjednostavniji izbor je $1/k$, ali se oni mogu odrediti i na drugačije načine [71]. Ova metoda se može koristiti bilo za regresiju ili klasifikaciju (više detalja o načinu korišćenja ove metode dato je u poglavlju 3.3).

Metoda podržavajućih vektora (eng. *support vector machine*) [38, 21], skraćeno SVM, je u svojoj prvobitnoj formi namenjena za binarnu klasifikaciju. Metoda konstruiše hiperravan koja deli trening skup u dve klase tako što maksimizuje *marginu* između najbližih tačaka (instanci) tih klasa. Tačke koje se nalaze na granici margine se zovu *podržavajući vektori*, a sredina margine je hiperravan koju određuje metoda SVM (slika 3.1). U opštem slučaju, veća margina povlači i manju grešku klasifikatora. Tačkama koje se nalaze sa pogrešnih strana hiperravni se dodeljuju težine koje odgovaraju udaljenostima tih tačaka od hiperravni. Potrebno je izračunati minimalnu vrednost funkcije koja kao argumente ima ove težine, uzimajući u obzir različite moguće hiperravni. Hiperravan sa minimalnom vrednošću funkcije određuje marginu (ona minimizuje grešku prouzrokovanu tačkama sa pogrešnih strana hiperravni). Kada nije moguće naći linearno razdvajanje, tačke se obično projektuju u više-dimenzioni prostor, gde ih je moguće linearno odvojiti (ovo se po-

stiže pomoću *kernel* tehnika). *Regresija podržavajućih vektora* (eng. *support vector regression*) [116], skraćeno SVR, predstavlja modifikaciju metode SVM i prilagođena je rešavanju problema regresije.



Slika 3.1: Klasifikacija pomoću SVM.

Metoda *slučajnih šuma* (eng. *random forests*) [26] koristi stabla odlučivanja i može se koristiti i za klasifikaciju i za regresiju. Svako stablo odlučivanja se konstruiše na osnovu slučajno odabranog podskupa trening podataka i u konstrukciji čvorova stabla se razmatraju slučajno odabrani podskupovi atributa. U čvorovima stabala se na osnovu vrednosti atributa vrši grananje. Kada je potrebno izvršiti klasifikaciju/regresiju, vrši se kretanje kroz svako stablo počev od korena do nekog lista. U svakom čvoru, atributi instance se koriste u odabiru grane kojom će se dalje nastaviti, sve dok se ne dođe do lista. Svako od stabala vrši klasifikaciju/regresiju i agregacijom se određuje izbor cele šume.

Veštačke neuronske mreže (eng. *artificial neural networks*) [95] su zasnovane na biološkim neuronskim mrežama, tj. na centralnom nervnom sistemu čoveka. Sastoje se iz čvorova koji imitiraju neurone čovekovog mozga. Čvorovi su spojeni vezama i međusobno komuniciraju. Mogu primati informacije, obrađivati ih i prosleđivati drugim čvorovima. Čvorovi su obično organizovani po nivoima. Ulazni nivo prima informacije, komunicira sa drugim nivoima i na kraju je rezultat dostupan u izlaznom nivou.

3.2 Pregled postojećih portfolija za rešavanje problema SAT i CSP

Detaljan pregled tehnika izbora algoritama napravio je Kothof [78]. Ovde kratko opisujemo samo najpoznatije pristupe.

3.2.1 SAT portfoliji

Problem automatskog odabira rešavača za instancu koju treba rešiti je dosta proučavan problem u SAT zajednici. Na osnovu atributa ulazne instance, ili se podešavaju neki parametri fiksiranog rešavača (u opštem slučaju u pitanju je problem konfigurisanja algoritma), ili se jedan od nekoliko dostupnih rešavača bira za rešavanje te instance (u opštem slučaju u pitanju je problem izbora algoritma). Najuspešniji pristupi su zasnovani na tehnikama mašinskog učenja. Svaka SAT instanca se karakteriše skupom njenih atributa. Korpus instanci se rešava pomoću različitih SAT rešavača (ili pomoću jednog rešavača konfigurisanog sa različitim kombinacijama parametara) i za svaku instancu su vremena njenog rešavanja pomoću svakog od rešavača pridružena vektoru atributa te instance. Jedan mogući pristup je da se za svaki rešavač konstruiše prediktivni model gde on opisuje zavisnost između vektora atributa i vremena rešavanja. Kada je potrebno rešiti novu instancu, na osnovu vektora atributa te instance, model svakog rešavača predviđa njegovo vreme rešavanja i bira se rešavač sa najmanjim predviđenim vremenom.

Atributi korišćeni u SAT portfolijima [101, 134] se obično mogu podeliti u nekoliko grupa. Neformalno ćemo opisati jednu moguću podelu u grupe.

U prvu grupu atributa spadaju broj klauza c u ulaznoj formuli, broj promenljivih v, njihov odnos c/v, procenat binarnih klauza, procenat ternarnih klauza (klauza od 3 literala), itd. Da bi se odredili atributi druge grupe, prvo se konstruiše graf koji za svaku promenljivu i klauzu sadrži po čvor kao i grane koje povezuju promenljive i klauze koje ih sadrže. Iz ovog grafa se potom određuju stepeni čvorova koji odgovaraju promenljivama i stepeni koji odgovaraju klauzama. U treću grupu spadaju razni odnosi: odnos pozitivnih i negativnih literala u svakoj od klauza, odnos pozitivnih i negativnih pojavljivanja svake od promenljivih, odnos binarnih i ternarnih klauza, itd. Ostale grupe se mogu odnositi na atribute koji se dobijaju iz kratkih pokretanja SAT rešavača (npr. broj jediničhnih propagiranja klauza, procena veličine prostora pretrage, atributi koje se dobijaju iz SAT rešavanja putem lokalne

pretrage, itd). Ukupan broj atributa može biti veliki (preko 100) i detaljniji njihov prikaz se može pronaći u pomenutoj literaturi. U nastavku ćemo ukratko opisati najpoznatije SAT portfolije.

`SATzilla` [134] je automatski pristup konstrukciji portfolija za SAT koji koristi *empirijski model težine* (eng. *empirical hardness model*) za predviđanje vremena rešavanja. Rešavači zasnovani na ovom pristupu i razvijeni od istih autora su više puta pobeđivali na SAT takmičenjima (na primer, 2007. i 2009. godine). `SATzilla` je verovatno presudno uticala na izuzetan napredak u razvoju velikog broja efikasnih portfolio pristupa. Ipak, ovaj pristup ima i mane. Teško je razumeti kako funkcioniše, a kompleksnost pristupa utiče i na to da reimplementacija ovog sistema bude vrlo težak i zahtevan zadatak.

Postoji više verzija ovog portfolija, a mi opisujemo onu koja je presudno doprinela njegovom uspehu. `SATzilla` prvo pokreće dva fiksirana rešavača sa veoma kratkim vremenskim ograničenjima sa ciljem brzog rešavanja lakih instanci. Ako instanca nije rešena, razmatra se izračunavanje njenih atributa. Izračunavanje atributa može dosta da potraje, pa se prvo predviđa potrebno vreme za dobijanje atributa pomoću empirijskog modela težine. Ako je predviđeno vreme veće od 2 minuta, pokreće se podrazumevani rešavač. U suprotnom se izračunavaju atributi, predviđa vreme rešavanja za svaki od rešavača i pokreće rešavač sa najmanjim predviđenim vremenom. U slučaju da on iz nekog razloga prestane da radi (na primer, zato što počne da koristi previše memorije), pokreće se sledeći najbolji rešavač.

Novi `SATzilla` sistem je izgrađen za svaku kategoriju instanci (na SAT takmičenjima). Sa ciljem predviđanja vremena rešavanja, empirijski model težine se trenira za svaki od ovih sistema i za svaki od rešavača. Predviđanje se vrši kombinovanjem predviđanja odvojenih modela za zadovoljive i nezadovoljive instance. Da bi ovo bilo omogućeno, koristi se procena verovatnoće da li je instanca zadovoljiva. Ova procena se dobija pomoću logističke regresije. Vrši se procena efikasnosti atributa i na osnovu nje bira se podskup skupa atributa koji će se koristiti. Ridž (eng. ridge) regresija [21] se koristi za predviđanje vremena rešavanja na osnovu izabranih atributa. Novije verzije portfolija `SATzilla` koriste i nove metode mašinskog učenja, ali ovde izbegavamo navođenje tih detalja.

ArgoSmArT [101] je sistem razvijen za odabir politika fiksiranog SAT rešavača pri rešavanju ulazne instance. Kao fiskirani rešavač se koristi ArgoSAT [90]. Sve instance trening skupa su podeljene u klase. Za instancu koju treba rešiti se izračunava podskup atributa koje koristi `SATzilla`, nalazi se najbliži sused iz trening

skupa i određuje kojoj klasi c pripada. Potom se instanca rešava pomoću konfiguracije rešavača za koju se zna da ima najbolje performanse na klasi c. Sistem ArgoSmArT podrazumeva da su dobre konfiguracije rešavača unapred poznate.

ArgoSmArT k-NN [102] je nadogradnja sistema ArgoSmArT i evaluiran je u odabiru rešavača, a ne konfiguracija kao originalni sistem. Koristi se algoritam *k-NN* za nalaženje k najbližih suseda ulazne instance. Na instanci se pokreće rešavač koji ima najbolje performanse na susedskim instancama. Jedna od osnovnih ideja ovog pristupa je jednostavnost: za razliku od većine drugih sistema koji su vrlo kompleksni i nepogodni za implementaciju, ArgoSmArT k-NN je vrlo lako implementirati. Uprkos jednostavnosti, pokazano je da je ovaj sistem efikasniji od portfolija `SATzilla` na korpusima korišćenim na zvaničnim takmičenjima SAT rešavača.

`ISAC` [89] je konfigurator rešavača koji koristi `SATzilla` atribute. Trening instance se dele u familije (klastere) koristeći tehnike klasterovanja [21]. Izračunavaju se atributi test instance i nalazi najbliži centar klastera među klasterima koji su na raspolaganju. Ako je udaljenost do tog centra manja od neke fiksirane granice, najbolja konfiguracija tog klastera se koristi za rešavanje. Inače, koristi se konfiguracija koja daje najbolje rezultate na celom trening korpusu.

`EISAC` [85] je nadogradnja portfolija `ISAC` (`Evolving ISAC`). Autori predstavljaju portfolio koji se menja (eng. evolve) tokom vremena, sa novim instancama koje postaju dostupne tokom vremena i koje se rešavaju. Treniranje portfolija samo jednom može dovesti do loših performansi sa pojavom novih problema koje treba rešiti. Instance koje rešava tokom faze eksploatacije `EISAC` koristi da povremeno izvrši ponovno treniranje. Ovime se portfolio prilagođava potencijalno novim problemima, a takođe koristi i nove instance da bi unapredio efikasnost. U ponovnom treniranju se mogu koristiti nove instance, novi atributi i novi rešavači. Pre ponovnog treniranja se vrši procena da li je ono neophodno, i samo u određenim (relativno retkim) slučajevima se ono ponovo i izvršava. Eksperimenti pokazuju da kada su instance za treniranje reprezentativne, tj. među njima ima instanci svih problema ulaznih instanci, sa pojavom novih instanci `EISAC` ne unapređuje performanse portfolija `ISAC` [85]. Međutim, kada nove instance dolaze iz novih problema, onda se značajno unapređuje efikasnost potencijalnim uključivanjem i ovih instanci u trening skup. U slučaju kada su instance podeljene tako da su na početku dostupne najlakše a vremenom dolaze sve teže i teže instance, `EISAC` takođe značajno nadmašuje u efikasnosti `ISAC`.

Razvijen je i pristup koji koristi koriste statističke modele ponašanja (modele klasifikacije zasnovane na latentnim promenljivama) [113] rešavača za izbor algoritma. Predloženi modeli pokušavaju da obuhvate zakonitosti između rešavača, problema i vremena rešavanja. Svaka instanca trening skupa se rešava veliki broj puta pomoću svakog od rešavača i model se pravi da odgovara rezultatima trening faze korišćenjem iterativnog algoritma za maksimizaciju očekivanja. Tokom testiranja se model nadograđuje novim rezultatima. Pri izboru se bira rešavač i vreme za koje će biti pokrenut korišćenjem procene korisnosti rešavača na pokretanju sa velikim vremenskim ograničenjem.

3S [87] je portfolio rešavač koji je učestvovao i pobeđivao na SAT takmičenjima. Prvih 10% vremena se pokreće fiksirani redosled rešavača od kojih svaki ima na raspolaganju određeno vreme za rešavanje. U slučaju da instanca nije rešena pomoću nekog rešavača u fiksiranom redosledu, izabranom rešavaču se dodeljuje preostalih 90% vremena. Fiksirani redosled rešavača se određuje na osnovu rezultata treninga. Princip izbora rešavača je zasnovan na algoritmu *k-NN* i vrlo je sličan portfoliju ArgoSmArT k-NN.

Malicki i koautori [86] su razvili portfolio sličan portfoliju ArgoSmArT k-NN i to u približno istom vremenskom periodu. Autori koriste algoritam *k-NN*, atribute sistema `SATzilla` i euklidsku udaljenost. Pokazano je da je konstruisani portfolio efikasniji od portfolija `SATzilla`.

Malicki i koautori [71] su nadogradili pristup opisan u prethodnom pasusu tako da koristi algoritam *k-NN* kod koga su susedima dodeljene težine – što je neka instanca bliža instanci koju je potrebno rešiti, to ima i veću težinu, pa se time rešavači pogodni za bliže instance favorizuju u izboru. Još jedno unapređenje u osnosu na originalni pristup je i korišćenje podele u klastere instanci u trening skupu – svakom klasteru se pridružuje broj suseda k najpogodniji za taj klaster. Kada je potrebno rešiti ulaznu instancu, određuje se kom klasteru pripada ta instanca i bira broj k koji je pridružen tom klasteru. Konačno, autori uvode i sekvencijalno pokretanje većeg broja rešavača na ulaznoj instanci, pri čemu je veliki deo rada posvećen postupku odabira ovih rešavača. Sva uvedena unapređenja omogućavaju da novi portfolio neznatno nadmaši originalni *k-NN* u efikasnosti.

Pristup *Hijerarhijsko klasterovanje osetljivo na cenu* (eng. *Cost-Sensitive Hierarchical Clustering*) [88] je razvijen ca ciljem da se dobije portfolio pogodan za različite domene. Ovaj portfolio u fazi treniranja počinje od jednog klastera u koji

su smeštene sve instance. Klaster se deli na sve manje i manje klastere, pri čemu se deoba vrši tako da se instance novih klastera što je više moguće "slažu" u tome koji je rešavač najpogodniji za nove klastere. Za procenu pogodnosti rešavača na skupu instanci koristi se *k-NN* algoritam, jer je on brz i može se koristiti u raznim domenima. Klasteri se smanjuju do neke minimalne veličine (npr. 10 instanci). Pokazuje se da je ovaj pristup konkurentan portfolijima `SATzilla` i 3S na problemima SAT i MaxSAT.

3.2.2 CSP portfoliji

Portfoliji su primenjeni i na rešavanje problema CSP, ali u značajno manjoj meri nego što je to slučaj kod problema SAT. Veliki broj ideja u oblasti SAT portfolija iskorišćen je i kod CSP portfolija, a više puta su SAT portfoliji prilagođavani problemima CSP. To se najlakše postiže korišćenjem istog algoritma kao i u slučaju SAT portfolija, ali sa atributima koji su karakteristični za CSP instance i uz odabir CSP (umesto SAT) rešavača u rešavanju CSP (umesto SAT) instanci. Ovakav način prilagođavanja portfolija iz jedne oblasti drugoj je relativno lako izvodljiv, pa više oblasti obično ima korist iz razvoja efikasnih portfolija u bilo kojoj od tih oblasti (spomenute biblioteke ASlib i AClib kao jedan od ciljeva imaju upravo sakupljanje portfolija iz raznih oblasti na jednom mestu).

CPHYDRA [104] je portfolio za CSP koji koristi algoritam *k-NN* da odredi jedan ili više rešavača koji će biti pokrenut/pokrenuti na novoj CSP instanci. CPHYDRA koristi 36 CSP atributa. Superiornost ovog portfolija u odnosu na rešavače nad kojima je izgrađen se pokazuje korišćenjem izazovnih benčmark instanci sa CSP takmičenja. Glavna karakteristika ovog portfolija koja je iskorišćena u mnogim kasnijim pristupima je sekvencijalno korišćenje više od jednog potencijalnog rešavača. Sistem CPHYDRA je pobedio na takmičenju CSP rešavača koje je organizovano 2008. godine.

Još jedan pristup, opisan od strane Kiziltana i koautora [75], koristi klasifikatore vremena rešavanja (kategorije su: "kratko", "srednje" i "dugo") da minimizuje prosečno vreme rešavanja svake instance. Ovaj portfolio koristi atribute portfolija CPHYDRA i `SATzilla` i njihovu kombinaciju.

Amadini i koautori [2] su na problemima CSP poredili efikasnost različitih portfolija zasnovanih na SAT portfolio tehnikama i algoritmima mašinskog učenja. Re-

zultati pokazuju da savremeni SAT portfolio pristupi ostvaruju najbolje performanse i na polju rešavanja problema CSP.

Skoriji sistem `SUNNY` [3] po efikasnosti nadmašuje pristup CPHYDRA kao i neke portfolije razvijene za SAT, ali prilagođene CSP rešavanju (na primer, `SATzilla`). Za neku novu instancu koju treba rešiti, `SUNNY` koristi algoritam *k-NN* da izabere jedan ili više rešavača koje treba pokrenuti. U praksi, broj izabranih rešavača je vrlo retko veći od 2. Izabrani rešavači dele dostupno vreme (svakom izabranom rešavaču se dodeljuje određena količina vremena iz dostupnog vremena, na osnovu efikasnosti tog rešavača na k najbližih instanci). Metodologija za odabir k nije predstavljena, a najbolja vrednost ovog parametra se izračunava izvršavanjem unakrsne provere za razne vrednosti k. `SUNNY` koristi 155 atributa, dobijenih iz instanci u MiniZinc ulaznom formatu, korišćenjem programa *mzn2feat* koji je razvijen od strane istih autora. Od ovih atributa, 11 su dinamički, tj. dobijeni pokretanjem rešavača Gecode sa vremenskim ograničenjem od 2 sekunde. Originalna eksperimentalna evaluacija [3] uključuje instance sa CSP takmičenja i iz MiniZinc distribucije i 11 rešavača koji su učestvovali na takmičenju MiniZinc Challenge.

`Proteus` [63] je hijerarhijski portfolio koji je po efikasnosti nadmašio mnoge druge pristupe mašinskog učenja prilagođene CSP rešavanju. Kada je potrebno rešiti novu instancu, rešavač se bira na osnovu odluka koje se donose na dva ili tri nivoa. Na prvom nivou bira se samo pristup rešavanju CSP instance – ili CSP rešavanje zasnovano na pretrazi i propagiranju ili svođenje na SAT. Ako je izabrano rešavanje zasnovano na pretrazi i propagiranju, onda se na drugom nivou bira rešavač koji će biti pokrenut. Ako je izabrano svođenje na SAT, onda se na drugom nivou bira kodiranje, a na trećem nivou SAT rešavač koji će biti pokrenut. Kako autori navode, prednost ovog pristupa je da se na svakom nivou može iskoristiti najpogodnija tehnika za taj nivo, što može dovesti do poboljšanja performansi. Sa druge strane, korišćenjem različitih tehnika na svakom nivou utiče na značajno veću kompleksnost sistema u odnosu na mnoge druge pristupe. `Proteus` koristi iste atribute kao i CPHYDRA, a među ovim atributima neki su dinamički, tj. dobijene pokretanjem rešavača *Mistral* sa vremenenskim ograničenjem od 2 sekunde. Za svako od 3 SAT kodiranja, `Proteus` koristi 54 SAT atributa. Instance sa CSP takmičenja su korišćene i `Proteus` je testiran sa 4 CSP rešavača, 3 SAT kodiranja i 4 SAT rešavača.

3.3 CSP Portfolio ArgoCSP-kNN

U ovom poglavlju predstavljamo portfolio koji smo razvili za probleme CSP. Koristimo metodu k-najbližih suseda pri čemu je princip funkcionisanja isti kao kod SAT portfolija ArgoSmArT k-NN [102], uz izmene da je novi portfolio konstruisan sa atributima i rešavačima prilagođenim za probleme CSP.

3.3.1 Atributi portfolija

Za razliku od nekih drugih pristupa (na primer, [63, 75]) koji koriste atribute generisanih SAT instanci (u slučaju svođenja na SAT), mi koristimo samo one dobijene iz ulazne CSP instance. Koristili smo 70 atributa podeljenih u nekoliko grupa: atributi koji se odnose na veličine domena svih celobrojnih promenljivih (na primer, prosečna veličina domena), atributi koji se odnose na broj svih promenljivih i na promenljive sa nekontinuiranim domenima, atributi koji se odnose na broj i procenat ograničenja različitih tipova — intenzionalna ograničenja (na primer, procenat intenzionalnih ograničenja među svim ograničenjima), ekstenzionalna ograničenja, globalna ograničenja (na primer, prosečna arnost globalnih ograničenja). Uključeni su i atributi koji se odnose na specifične tipove ograničenja (na primer, broj aritmetičkih operacija, broj množenja, suma domena promenljivih uključenih u množenja, broj *all-different* ograničenja), itd. Spisak svih atributa dostupan je na veb-strani navedenoj u uvodu ove teze.

Primer 16. *Dajemo jednostavan primer CSP instance i izračunavamo neke od njenih atributa.*

```
(int x1 0 3)
(int x2 0 4)
(int x3 1 5)
(alldifferent x1 x2 x3)
(<= 6 (+ x1 x2))
```

Suma veličina domena promenljivih uključenih u sabiranje ili oduzimanje je 9 (`x1` *može uzeti 4 i* `x2` *može uzeti 5 vrednosti). Broj pojavljivanja globalnih ograničenja je 1 (pojavljivanje ograničenja all-different). Broj pojavljivanja intenzionalnih ograničenja je 1 (jedna nejednakost). Prosečna arnost globalnih ograničenja je 3 (jedino globalno ograničenje ima 3 argumenta). Procenat globalnih ograničenja posmatrajući sva ograničenja je 50%. Broj aritmetičkih operacija je 1.*

U većini eksperimenata – izuzev kada smo eksplicitno naglasili da je rađeno drugačije – smo koristili svih 70 atributa. Vreme potrebno za dobijanje atributa je vrlo malo (na računaru koji smo koristili iznosilo je prosečno 0.05 sekundi na svima instancama korišćenim u eksperimentima).

3.3.2 Opis portfolija

Pre nego što se može primeniti, portfolio ArgoCSP-kNN mora biti istreniran. *Treniranje portfolija* se sastoji iz faze rešavanja trening korpusa svim dostupnim rešavačima (ovu fazu zovemo *priprema*) i faze *treniranje prediktivnog modela*. Jednom istreniran, portfolio se može primeniti mnogo puta. Fazu primene zovemo *primena napravljenih modela na nepoznate podatke*, kraće, *eksploatacija* ili *testiranje*.

Priprema. Na početku se prikupljaju atributi svih trening instanci kao i performanse svih rešavača na tim instancama. Za rešavanje svake trening instance, svakom rešavaču na raspolaganju je određeno vreme, koje zovemo *vremensko ograničenje* (eng. *time limit*). Performanse rešavača na instanci su izražene pomoću *PAR10* skora [65] koji odgovara vremenu rešavanja ukoliko je instanca rešena u okviru datog vremenskog ograničenja, ili vremenskom ograničenju pomnoženom sa 10 u suprotnom. Rezultati faze pripreme su predstavljeni dvema tabelama. Za svaku instancu, prva tabela sadrži atribute. Druga tabela sadrži PAR10 skor za dato vremensko ograničenje za svaki rešavač i svaku instancu.

Treniranje prediktivnog modela. Cilj ove faze je da se izaberu (na osnovu rezultata faze pripreme) optimalne vrednosti metaparametara k (broj suseda) i d (funkcija rastojanja) koji će kasnije biti korišćeni u fazi eksploatacije. Ovo se postiže korišćenjem unakrsne provere. Isprobavaju se različite vrednosti k i d. Za svaku fiksiranu kombinaciju ovih metaparametara, izvršava se unakrsna provera sa 5 delova na pripremljenim podacima. U svakom krugu, za svaku instancu jednog dela nalazi se k najbližih suseda (koji se izračunavaju korišćenjem funkcije udaljenosti d) među instancama ostalih delova. Za instancu se bira rešavač sa najmanjom sumom PAR10 skorova na njenim susedskim instancama. Kombinacija k i d koja daje najbolje rezultate na svim trening instancama (najmanju sumu PAR10 skorova izabranih rešavača na svim trening instancama) je ona koju će portfolio koristiti u fazi eksploatacije.

Portfolio i njegova eksploatacija. Algoritam izbora rešavača zasnovan na portfoliju ArgoSmArT [102] je prikazan na slici 3.2. Tokom pripreme se sakupljaju vrednosti argumenta `pripremni_podaci`, dok se vrednosti ulaznih parametara `k` i `d` biraju tokom treninga prediktivnog modela. U fazi eksploatacije se prvo izračunavaju atributi ulazne instance. Potom se korišćenjem vrednosti `d` nalazi `k` najbližih suseda ulazne instance na celom trening korpusu i najbolji rešavač na ovim instancama se koristi da reši ulaznu instancu. Funkcija `Par10` vraća sumu PAR10 skorova nekog konkretnog rešavača na susedskim instancama.

`instanca` – ulazna instanca koju treba rešiti
`rešavači` – skup rešavača
`pripremni_podaci` – atributi i PAR10 skorovi rešavača na trening instancama
`k` – broj susedskih instanci
`d` – funkcija udaljenosti

```
function ArgoCSP-kNN (instanca, rešavači, pripremni_podaci, k, d)
  atributi = CSP_atributi (instanca);
  susedi = Najbliži_susedi (pripremni_podaci, k, d, atributi);
  najbolji_skor = ∞;
  for s in rešavači
    if (Par10 (s, susedi) < najbolji_skor)
      najbolji_skor = Par10 (s, susedi);
      najbolji_rešavač = s;
  return najbolji_rešavač;
```

Slika 3.2: Procedura izbora portfolija ArgoCSP-kNN.

3.4 Upotreba portfolija za odabir kodiranja u okviru sistema *meSAT*

U poglavlju 2.5.2 je pokazano da su različita kodiranja pogodna za različite probleme, pa postoji motivacija za upotrebu portfolija za automatski odabir kodiranja. Sa ciljem da pokažemo prednosti automatskog odabira, sproveli smo eksperimentalnu evaluaciju. Instance, eksperimentalno okruženje i korišćena kodiranja su isti kao u pomenutom poglavlju.

Primenili smo opisani ArgoCSP-kNN pristup i poredili ga sa (i) *orakl pristupom* (ili virtuelno najboljim pristupom) koji bi odabrao najbolje kodiranje za svaku instancu (ovaj metod nije moguć u praksi jer pravi savršene odluke i za svaku instancu

pogađa optimalno kodiranje pre pokušaja rešavanja instance, što je nemoguće implementirati) i (ii) sa *najboljim fiksiranim* metodom – kodiranje koje daje ukupno najbolje rezultate. Naglasimo da se u pseudokodu prikazanom na slici 3.2 bira se između raznih rešavača, dok u ovom poglavlju izbor vršimo između različitih kodiranja jednog rešavača (*meSAT*). Za potrebe prilagođavanja pomenutom pseudokodu, može se smatrati da je *meSAT* sa svakim pridruženim kodiranjem poseban rešavač.

Izbor je načinjen među tri tipa kodiranja: kodiranja direktno-podrška, kodiranja uređenja i kodiranja direktno-uređenje. Kao što je već rečeno, kodiranje direktno-uređenje je neznatno efikasnije od direktnog kodiranja i kodiranja podrške pa ova dva kodiranja ne koristimo. U pripremnoj fazi su sve instance rešavane pomoću sve tri vrste kodiranja i sakupljeni su atributi ovih instanci. U fazi treniranja prediktivnog modela su isprobane sve moguće kombinacije različitih vrednosti k (koje je uzimalo vrednosti od 1 do 20) i 4 različitih funkcija udaljenosti ([130]):

$$d_1(X,Y) = \sqrt{\sum_i (x_i - y_i)^2} \quad d_2(X,Y) = \sum_i \left(\frac{x_i - y_i}{\sqrt{|x_i y_i| + 1}} \right)^2$$

$$d_3(X,Y) = \sum_i \frac{|x_i - y_i|}{\sqrt{|x_i y_i|} + 1} \quad d_4(X,Y) = \sum_i \left(\frac{x_i - y_i}{\sqrt{|x_i y_i|} + 10} \right)^2$$

Optimalne vrednosti ovih metaparametara su korišćene od strane portfolija. Portfolio je u slučaju jednakih najboljih skorova najveći prioritet davao kodiranju direktno-uređenje, a najmanji kodiranju uređenja.

Rezultati evaluacije su dati u tabeli 3.1 (za sve instance) i tabeli 3.2 (za interesantne instance). Za procenu efikasnosti portfolija korišćena su dva pristupa: već pomenuta unakrsna provera sa 5 delova (metod označen sa $auto_{cv}$ u tabeli) i treniranje samo na lakim instancama i testiranje na ostatku (metod označen sa $auto_{easy}$ u tabeli).

Unakrsna provera. Tabela 3.1 pokazuje da je metod $auto_{cv}$ omogućio da *meSAT* reši 33 instanci više nego korišćenjem najboljeg fiksiranog kodiranja. Iako ovo možda ne deluje ubedljivo kada se posmatraju svih 1379 instanci, rezultati su takvi zbog velikog broja instanci rešenih od strane svih kodiranja (889) i broja teških instanci koje nisu rešene ni jednim kodiranjem (295). Ako posmatramo samo rezultate na interesantnim instancama (koje jedino mogu da utiču na ukupan broj rešenih instanci), dobijamo malo jasniju sliku. Tabela 3.2 pokazuje da *najbolji-fiksirani* metod (kodiranje direktno-uređenje) rešava 164 instanci, da $auto_{cv}$ rešava 197, a da *orakl*

rešava 208 instanci. Takođe, ukupno vreme rešavanja je skoro prepolovljeno (od 763 na 419 minuta). Ovo se može smatrati značajnim unapređenjem. Najveće unapređenje je postignuto na MiniZinc korpusu, dok je na CPAI09 korpusu unapređenje malo (jer je $auto_{cv}$ rešio isti broj instanci kao i *najbolji-fiksirani* metod, ali za nešto kraće vreme).

Tabela 3.1: Rezultati eksperimenalne evaluacije zasnovane na rezultatima predstavljenim u tabeli 2.2. Dodatak su metode auto – odabir kodiranja pomoću portfolija ArgoCSP-kNN: $auto_{easy}$ – treniranje na lakim instancama, $auto_{cv}$ – unakrsna provera, *orakl* – najbolje kodiranje za instancu.

		meSAT						Sugar	Azucar		mzn-g12	
korpus	#	ds	o	do	$auto_{cv}$	$auto_{easy}$	orakl	Sugar	m2	log	cpx	lazy
CPAI09	583	359(2402)	488(1099)	486(1104)	486 (1094)	487(1103)	492(1055)	**494(1053)**	448(1503)	428(1722)	332(2609)	348(2638)
MiniZinc	770	583(2100)	562(2516)	579(2266)	**609(1852)**	599(1940)	612(1811)	554 (2572)	509(2965)	486(3309)	371(4170)	492(2974)
D. kraljice	26	**23 (76)**	2 (240)	20 (89)	**23 (76)**	**23 (76)**	25 (60)	3 (236)	2 (246)	2 (247)	4 (224)	5 (211)
Ukupno	1379	965(4578)	1052(3855)	1085(3459)	**1118(3023)**	1109(3120)	1129(2926)	1051 (3862)	959(4713)	916(5278)	707(7002)	845(5824)

Tabela 3.2: Eksperimentalna evaluacija odabira kodiranja na interesantnim instancama. Korišćene metode rešavanja su iste kao i u tabeli 3.1.

		meSAT						Sugar	Azucar		mzn-g12	
problem	#	ds	o	do	*$auto_{cv}$*	*$auto_{easy}$*	*orakl*	Sugar	m2	log	cpx	lazy
Ukupno	208	76(1507)	131(945)	164(763)	**197(419)**	188(518)	208(348)	142(849)	116(1057)	99(1263)	98(1165)	96(1255)

Treniranje na lakim instancama. U drugom pristupu smo želeli da ispitamo mogućnost smanjenja ukupnog vremena treniranja portfolija (koje se u slučaju eksperimenata u ovom poglavlju meri procesorskim danima pri treniranju na svim vrstama instanci) tako što će se treniranje vršiti samo na lakim instancama a testiranje na celom korpusu.

Lake instance nisu unapred poznate, pa se priprema sastojala od pokretanja sistema *meSAT* sa vremenskim ograničenjem od 5 sekundi na svim instancama korišćenjem svakog od 3 kodiranja. Od 1379 instanci, 543 su rešene pomoću svih kodiranja u okviru datog vremenskog ograničenja – ove instance ćemo smatrati *lakim instancama* i one su korišćene za trening. Pošto je vremensko ograničenje malo, maksimalno moguće vreme za pripremu je samo $1379 \times 3 \times 5$ sekundi, tj. samo oko 6 sati i moguće ga je izvršiti na standardnom PC računaru (u realnosti, bilo je potrebno samo oko 3.5 sati, pošto su mnoge instance rešene za manje od 5 sekundi).

Rešavanje 543 lake instance je urađeno za manje od 13 minuta, ali pošto ove instance nisu unapred poznate, kompletno vreme uključujući i rešavanje instanci koje ne pripadaju skupu lakih se mora uzeti u obzir.

Posle treniranja, faza eksploatacije je primenjena na celokupan korpus. Da bismo imali uniformni prikaz rezultata, prikazani rezultati u tabeli 3.1 i tabeli 3.2 su dati za ceo korpus (pa postoji preklapanje između trening i test skupa). Ipak, trening instance su tako lake da su rešene od strane svakog kodiranja; najbolje i najlošije kodiranje se na svim trening instancama ukupno razlikuju za manje od 5 minuta, a ovo vreme je zanemarljivo za ukupne rezultate (gde se vremena rešavanja kodiranja razlikuju u stotinama minuta). Takođe, pri razmatranju samo interesantnih instanci, ne postoji preklapanje između trening i test skupa.

Iako ne tako efikasno kao treniranje na svim instancama, treniranje samo na lakim instancama je takođe efikasnije u odnosu na *najbolji-fiksirani* metod. Pretpostavljamo da razlog uspehu ovom pristupu leži u prirodi razmatranih korpusa, u kojima su instance grupisane u familije (svaka familija potiče od jednog problema) i svaka familija sadrži vrlo slične instance koje se razlikuju samo u veličini, pri čemu optimalno kodiranje ne zavisi od veličine, nego i od strukture instance. Interesantno je da je glavna razlika u efikasnosti između metoda $auto_{easy}$ i $auto_{cv}$ napravljena na jednom problemu – M/latin_squares – na kome je $auto_{cv}$ rešio 10 instanci više. Drastična razlika u efikasnosti na tom problemu je nastala jer je najbolje kodiranje za problem M/latin_squares pri rešavanju sa kraćim vremenskim ograničenjem (180s) kodiranje direktno-uređenje, dok je najbolje kodiranje pri rešavanju sa većim vremenskim ograničenjem (600s) kodiranje uređenja. Ovo je bio jedini problem sa ovakvim "ponašanjem" na našem korpusu, pa pretpostavljamo da su ovakve vrste problema (gde treniranje samo na lakim instancama ne daje dobre rezultate) retke. Probali smo da povećamo vremensko ograničenje u fazi pripreme (inicijalno 5s) da bi uključili i nešto teže instance, ali je ovo produžilo fazu pripreme i ukupno vreme treniranja (što smo inicijalno i želeli da izbegnemo), a nije značajno unapredilo rezultate.

3.5 Korišćenje portfolija u odabiru rešavača

Pri rešavanju CSP instanci, portfoliji ne moraju praviti izbor samo između različitih SAT kodiranja, već i između različitih rešavača. U ovom poglavlju ćemo prvo izvršiti eksperimentalno poređenje CSP rešavača, a rezultati ovog poglavlja će

se koristiti u poglavljima koja slede. U drugom poglavlju poredimo efikasnost pristupa ArgoCSP-kNN sa različitim, već postojećim, portfolio pristupima. Na kraju, procenjujemo efekat vremena rešavanja instance na efikasnost portfolija. Rezultati prikazani u ostatku poglavlja su zajednički rezultati Mladena Nikolića, Filipa Marića i autora teze.

3.5.1 Eksperimentalna evaluacija rešavača

Koristili smo veliki broj dostupnih CSP rešavača i bogat korpus dostupnih CSP benčmark instanci. Pored opisivanja rešavača i korpusa, predstavljamo i evaluaciju tih rešavača na tim korpusima.

Rešavači. Za svođenje na SAT, koristili smo kodiranje direktno-podrška, kodiranje uređenja i kodiranje direktno-uređenje implementirane u sistemu *meSAT* 1.1, kodiranje uređenja implementirano u sistemu Sugar 2.1.3 [124], i logaritamsko kodiranje i kompaktno kodiranje uređenja implementirane u sistemu Azucar 0.2.4 [128]. SAT rešavač Minisat 2.2 [45] je korišćen u svim slučajevima kada je vršeno svođenje na SAT. SMT rešavači Yices 2.2.0 [44] i Z3 4.2 [41] su korišćeni za rešavanje generisanih SMT-LIB instanci. Eksperimenti su sprovedeni i za dva rešavača koji kombinuju pretragu i propagiranje ograničenja: *Abscon* 112V4 [93] i *Mistral* 1.545 [60]. Takođe smo evaluirali rešavače iz G12 MiniZinc 1.6 [98] distribucije: `mzn-g12lazy` i `mzn-g12cpx`, koji implementiraju lenjo generisanje klauza, kao i `mzn-g12fd` i `mzn-g12mip`. Rešavač Gecode 4.2.1 [110] je takođe korišćen.

Instance. Koristili smo dva javno dostupna korpusa CSP instanci: (i) CPAI09 korpus koji sadrži sve instance sa takmičenja Fourth International CSP Solver Competition[1], (ii) instance iz MiniZinc korpusa dostupne na veb-strani[2]. Upotrebili smo tri formata ulaznih datoteka: MiniZinc jezik, XCSP i Sugar jezik.

Instance su konvertovane između različitih jezika pomoću ranije predstavljene sheme na slici 2.4. Isključili smo instance koje nije bilo moguće konvertovati iz bilo kog od jezika u neki drugi jezik kao i instance za koje je postojao rešavač koji je davao pogrešnu zadovoljivost (iznenađujuće, bilo je 541 instanci na kojima je barem jedan od rešavača dao pogrešan odgovor, a 8 od 15 rešavača je dalo pogrešan odgovor na nekim instancama). Finalni korpus se sastojao od 8436 instanci.

[1] `http://www.cril.univ-artois.fr/CPAI09`
[2] `http://www.minizinc.org`

Eksperimentalno okruženje. Svi testovi su vršeni na istom računaru koji je već opisan u poglavlju 2.5.2. Vremensko ograničenje je 600 sekundi po instanci (osim na mestima gde je posebno naglašeno drugačije) i obuhvata odabir rešavača/kodiranja gde je to potrebno, samo kodiranje gde je to potrebno, i na kraju rešavanje.

Poređenje rešavača. Pokretali smo sve rešavače na svim instancama. Rezultati su prikazani u tabeli 3.3. Prikazani su rezultati i za dva dodatna metoda: *orakl* i *najbolji-fiksirani*, koji su već opisani u poglavlju 3.3. Rezultati pokazuju da je razlika između metoda *najbolji-fiksirani* i *orakl* 1415 instanci, pa postoji dobra motivacija za korišćenje portfolio pristupa.

Tabela 3.3: Rezultati eksperimentalne evaluacije CSP rešavača. Kodiranje korišćeno za svođenje na SAT je dato u zagradama, a skraćenice za kodiranja su ista kao i u tabeli 2.2 uz dodatak oznake za kompaktno kodiranje podrške (`co`) i oznake za logaritamsko kodiranje (`log`). Simbol # označava broj rešenih instanci. Vreme je dato u danima a za svaku nerešenu instancu, podrazumeva se 600 sekundi utrošenog vremena.

Metoda rešavanja	Rešavač	# (od 8436)	Vreme
Pretraga i propagiranje	*Abscon*	4402	30.2
	Mistral	6216	16.9
Svođenje na SAT	*meSAT* (ds)	3128	38.6
	meSAT (o)	3985	32.6
	meSAT (do)	3984	32.6
	Sugar (o)	5548	22.7
	Azucar (co)	4402	31.1
	Azucar (log)	3804	30.7
Svođenje na SMT	Z3	4938	26.7
	Yices	4978	26.0
Lenjo generisanje klauza	`mzn-g12cpx`	2883	39.4
	`mzn-g12lazy`	2865	41.3
Ostali	`mzn-g12mip`	1544	49.0
	`mzn-g12fd`	2716	41.7
	Gecode	2627	41.4
	najbolji-fiksirani	6216	16.9
	orakl	7631	7.0

3.5.2 Poređenje sa drugim CSP portfolijima

U ovom poglavlju poredimo ArgoCSP-kNN sa drugim CSP portfolijima. Pošto je najvažniji deo našeg rada metodologija zasnovana na vrlo kratkim vremenskim ograničenjima, a ne sam portfolio, naš cilj nije da ArgoCSP-kNN nadmaši u efikasnosti ostale pristupe, nego da pokažemo da je sa njima uporediv. Poređenje je izvršeno sa portfolijima `SUNNY` i `Proteus`, a ranije je pokazano da su oni u efikasnosti prevazišli mnoge druge razvijene pristupe.

Poređenje sa portfoliom `SUNNY`. ArgoCSP-kNN i `SUNNY` koriste različite rešavače, atribute i instance, pa je direktno poređenje dva pristupa bilo nemoguće bez adaptacije jednog od njih da koristi iste atribute, rešavače i instance kao i drugi. Implementirali smo portfolio `SUNNY` tako da koristi isti skup rešavača i atributa[3] kao i ArgoCSP-kNN i evaluirali ga na našem eksperimentalnom korpusu.

Pored različitog načina funkcionisanja, još jedna razlika između portfolija `SUNNY` i ArgoCSP-kNN je da `SUNNY` koristi fiksirani broj suseda `k` i neku fiksiranu funkciju udaljenosti `d`, dok ArgoCSP-kNN određuje optimalne vrednosti ovih metaparametara u fazi treniranja prediktivnog modela. Koristili smo broj suseda (`k` = 16) koji je dao najbolje rezultate u eksperimentima rada koji opisuje `SUNNY` kao i euklidsko rastojanje kao funkciju udaljenosti, pošto su njega koristili autori `SUNNY` pristupa. Za evaluaciju je korišćena unakrsna provera sa 5 delova. Rezultati poređenja našeg portfolija i portfolija `SUNNY` su dati u tabeli 3.4.

Tabela 3.4: Rezultati poređenja portfolija ArgoCSP-kNN i `SUNNY`.

	Broj rešenih instanci (od 8436)
najbolji-fiksirani	6216
`SUNNY`	7493
ArgoCSP-kNN	7511
orakl	7631

Rezultati pokazuju da je na korišćenom korpusu ArgoCSP-kNN uporediv sa portfoliom `SUNNY`. Malo unapređenje je možda dobijeno zbog fleksibilnosti našeg pristupa u izboru `k` i `d`, ali pošto unapređenje nije značajno, moguće je da bi situacija

[3]Takođe smo eksperimentisali sa programom `mzn2feat` za prikupljanje atributa, razvijenim od strane autora portfolija `SUNNY`, ali je program radio vrlo sporo na instancama korišćenih korpusa i nije mogao da prikupi atribute svih instanci čak ni posle nekoliko dana.

Tabela 3.5: Rezultati eksperimentalnog poređenja portfolija ArgoCSP-kNN i `Proteus`.

	Broj rešenih instanci (od 1493)
najbolji-fiksirani	984
ArgoCSP-kNN	1413
`Proteus`	1424
orakl	1493

bila obrnuta na drugačijem skupu instanci, pa zaključujemo da ne postoji značajna razlika u performansama.

Poređenje sa portfoliom `Proteus`. Pošto su svi podaci (atributi svih instanci kao i vremena rešavanja svih rešavača na tim instancama) korišćeni u radu u kome je opisan `Proteus` [63] javno dostupni, odlučili smo da primenimo ArgoCSP-kNN na ovim podacima. Sve instance tog korpusa zadovoljavaju dva uslova: nisu trivijalno rešene za 2 sekunde računanja atributa i rešene su od strane barem jednog od korišćenih rešavača tokom vremenskog ograničenja od 60 minuta. Korišćeno je 4 CSP rešavača i 6 SAT rešavača, kao i tri različita kodiranja (direktno kodiranje, kodiranje podrške i kodiranje uređenja) pri svođenju na SAT. U testiranju portfolija ArgoCSP-kNN nismo koristili atribute generisanih SAT instanci već samo 36 atributa dobijenih iz ulaznih CSP instanci. Rezultati poređenja između portfolija ArgoCSP-kNN i `Proteus` su dati u tabeli 3.5. Broj rešenih instanci za `Proteus` je preuzet iz rada koji uvodi ovaj portfolio. Kao i u slučaju poređenje sa portfoliom `SUNNY`, razlika je vrlo mala, pa možemo smatrati da dva portfolija ostvaruju približno iste performanse.

Primetimo da je ArgoCSP-kNN rešio redom 98.4% i 94.6% instanci od onih koje je mogao da reši u eksperimentalnim rezultatima predstavljenim u tabeli 3.4 i tabeli 3.5. Zbog bliskosti u efikasnosti sa metodom *orakl*, možemo smatrati sve evaluirane pristupe efikasnim savremenim pristupima.

3.5.3 Uticaj vremena rešavanja na efektivnost portfolija

Glavni deo vremena treniranja portfolija ArgoCSP-kNN se troši na fazu pripreme (rešavanje instanci), a ovo vreme je direktno zavisno od vremenskog ograničenja ko-

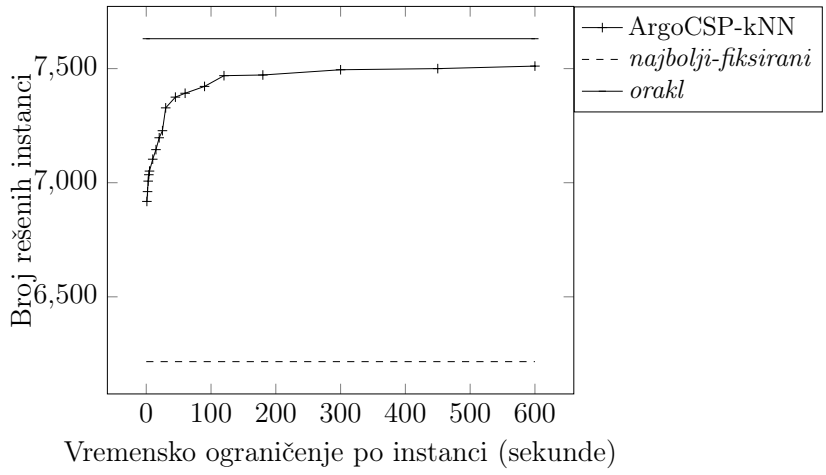
rišćenog u toj fazi. Pošto se ova faza izvršava samo jednom, obično smo spremni da dopustimo više vremena za pripremu da bi postigli bolje rezultate u fazi eksploatacije. Želimo da procenimo uticaj vremenskog ograničenja na ukupnu efikasnost portfolija ArgoCSP-kNN. U svim slučajevima, procena je rađena korišćenjem unakrsne provere sa 5 delova na celom korpusu. Faza pripreme je ponavljana više puta sa različitim vrednostima vremenskog ograničenja za rešavanje, od 1 do 600 sekundi, i da bi dobili fer poređenje, tokom unakrsne provere je korpus uvek deljen na istih 5 delova. Bez obzira na vremensko ograničenje korišćeno u fazi pripreme, vremensko ograničenje za odabrani rešavač u fazi eksploatacije je 600 sekundi.

Tabela 3.6: Rezultati eksperimentalne evaluacije portfolija ArgoCSP-kNN korišćenjem različitih vremenskih ograničenja (dato u sekundama). Ostala vremena data su u danima.

Vremensko ograničenje (sekunde)	1	5	10	30	60	90	120	300	600
Br. instanci rešenih u eksploataciji	6918	7051	7103	7328	7392	7422	7469	7495	7511
Vreme pripreme & treninga (dani)	1.4	5.8	10.8	28.9	53.8	77.2	99.7	219.3	393.2
Vreme eksploatacije (dani)	11.8	11.0	10.5	8.8	8.4	8.2	8.0	8.0	7.9
Ukupno vreme (dani)	13.2	16.8	21.3	37.7	62.2	85.4	107.7	227.2	401.2

Tabela 3.6 pokazuje dobijene rezultate i oni su predstavljeni grafikom na slici 3.3 (sa rezultatima dodatnih vrednosti vremenskog ograničenja). Slika jasno pokazuje da u početku postoji brzo povećanje u broju rešenih instanci sa povećanjem vremenskog ograničenja, ali da se nakon određenog trenutka kriva stabilizuje. Na primer, za vremenska ograničenja od 600 i 30 sekundi razlika u broju rešenih instanci nije velika (naročito kada se njihove performanse porede sa metodom *najbolji-fiksirani*), dok se vreme za pripremu i trening smanjuje sa 393.2 na 28.9 dana. Ovo pokazuje da je moguće postići skoro iste rezultate sa mnogo manjim vremenskim ograničenjima od 600 sekundi.

Primetimo da je treniranje sa vremenskim ograničenjem manjim od onog u fazi eksploatacije već korišćeno u treniranju na lakim instancama (metod $auto_{easy}$ u delu 3.4). Pristupi jesu vrlo slični, ali je razlika da se ovde u fazi eksploatacije koriste informacije i o onim instancama koje nisu rešene tokom faze treniranja, a ne samo o onima koje su rešene u toj fazi.



Slika 3.3: Rezultati eksperimentalne evaluacije portfolija ArgoCSP-kNN u poređenju sa metodama *najbolji-fiksirani* i *orakl*. Svaki znak na krivoj predstavlja jedan ishod zasnovan na rešavanju korpusa sa odgovarajućim vremenskim ograničenjem.

3.6 Portfolio pristup zasnovan na kratkom treniranju

Korisnici često za cilj imaju rešavanje što većeg broja instanci iz nekog fiksiranog skupa instanci, što je brže moguće, sa datim skupom dostupnih rešavača. Najjednostavniji način za postizanje dobrih rezultata je izbor jednog rešavača za koji se očekuje da će dati najbolje rezultate (na primer, rešavača koji je dao najbolje rezultate na nekom takmičenju) i potom njegova primena na svim instancama. Ipak, odabrani sistem možda nije efikasan u rešavanju specifičnog skupa instanci. Drugi pristup bi bio korišćenje prethodno treniranog portfolija. Međutim, taj portfolio je možda treniran na skupu instanci značajno različitim od onih koje je potrebno rešiti, pa se može desiti da bude vrlo neefikasan.

Ohrabreni rezultatima iz prethodnog poglavlja koji su pokazali da se može značajno smanjiti vremensko ograničenje bez značajnog smanjivanja ukupnog kvaliteta portfolija, predlažemo novi način efikasnog korišćenja portfolija u situacijama kada je potrebno korišćenjem skupa dostupnih rešavača što efikasnije rešiti korpus instanci takav da se prethodno nisu rešavale slične instance. Vršimo nadogradnju pristupa već opisanog u poglavlju 3.4 u kome smo trenirali samo na lakim instancama, i kao i u tom poglavlju, portfolio se trenira na istom skupu instanci koji treba rešiti. Zbog prirode zadatka (bitno je rešavanje samo jednog fiksiranog korpusa), generalizacija portfolija na instance van korpusa koji se rešava nije u prvom planu. Stoga pre-

klapanje između trening i test skupa instanci, koje se izbegava u evaluaciji tehnika mašinskog učenja, postaje nebitno u scenariju koji razmatramo.

Pristup je jednostavan i sastoji se u pokretanju svih rešavača na svim instancama korpusa sa datima kratkim vremenskim ograničenjem, vršenja treniranja prediktivnog modela, i potom primene portfolija na korpus koji sadrži samo instance koje nisu rešene tokom pripreme, pri čemu se izabranim rešavačima daje veće vremensko ograničenje od onog u fazi pripreme. Za razliku od treniranja samo na lakim instancama, ovde koristimo informacije o vremenima rešavanja na svim instancama, a ne samo na onima koje su lako rešive.

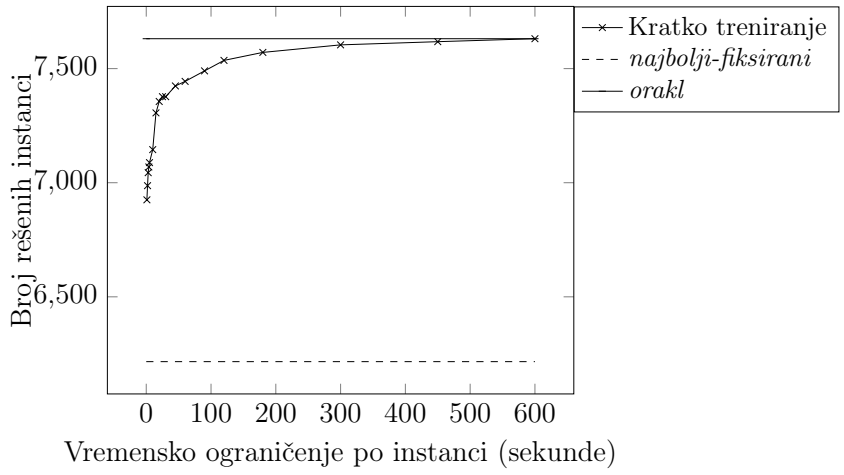
3.6.1 Evaluacija

U evaluaciji smo koristili razna vremenska ograničenja u fazi treniranja, ali zbog prirode našeg scenarija (ne možemo da dozvolimo velika vremenska ograničenja), fokusirali smo se na upotrebu vrlo malih vremenskih ograničenja. Rezultati su prikazani u tabeli 3.7 i predstavljeni grafikom na slici 3.4. Ukupan broj rešenih instanci naglo raste u regiji malih vremenskih ograničenja i onda se stabilizuje. Za vremensko ograničenje od 600 sekundi, nema potrebe pokretati odabrani rešavač na instancama nerešenim u fazi pripreme sa istim vremenskim ograničenjem (sve je već urađeno u fazi pripreme i broj rešenih instanci je isti kao i za rešavač *orakl*).

Tabela 3.7: Rezultati evaluacije portfolija ArgoCSP-kNN korišćenjem pristupa sa kratkim treniranjem za različite vrednosti vremenskog ograničenja (dato u sekundama). Ostala vremena data su u danima.

Vremensko ograničenje (sekunde)	1	5	10	30	60	90	120	300	600
Broj rešenih instanci	6925	7088	7145	7377	7444	7490	7536	7604	7631
Vreme pripreme & treniranja (dani)	1.4	5.8	10.8	28.9	53.8	77.2	99.7	219.3	393.2
Vreme eksploatacije (dani)	11.8	10.6	10.1	8.2	7.5	7.1	6.8	6.1	0
Ukupno vreme (dani)	13.2	16.4	20.9	37.1	61.3	84.3	106.5	225.4	393.2

Rezultati u tabelama 3.6 i 3.7 i krive na slikama 3.3 i 3.4 mogu delovati slično. Ključna razlika je da je evaluacija u tabeli 3.6 i slici 3.3 urađena korišćenjem unakrsne provere i u tom slučaju ne postoji preklapanje između trening i test skupa instanci. Suprotno tome, evaluacija čiji su rezultati prikazani u tabeli 3.7 i na slici 3.4 koristi isti korpus i za treniranje i za evaluaciju (ukoliko je instanca rešena tokom treniranja, nema potrebe rešavati je tokom testiranja). Zato, pristup koji

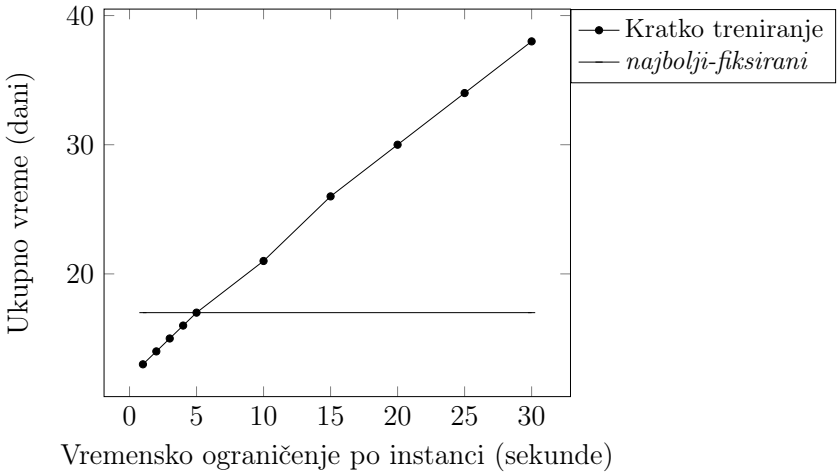


Slika 3.4: Poređenje broja rešenih instanci portfolija ArgoCSP-kNN korišćenjem pristupa sa kratkim treniranjem sa rešavačima *najbolji-fiksirani* i *orakl*. Svaki znak na krivoj predstavlja jedan ishod zasnovan na rešavanju korpusa sa odgovarajućim vremenskim ograničenjem.

koristi kratko treniranje konstantno postiže bolje performanse, ali ovo je i očekivano pošto se trening i test skup sastoje od istog skupa instanci.

U praktičnom scenariju koji smo opisali, centralne mere kvaliteta su broj rešenih instanci i ukupno vreme utrošeno za treniranje i evaluaciju – cilj je maksimizovati broj rešenih instanci uz utrošak što je moguće manje ukupnog vremena (na primer, tako da se i treniranje portfolija i njegova eksploatacija mogu izvesti na računaru koji korisnik ima na raspolaganju). Iznenađujuće, naši eksperimenti pokazuju da je čak i za vrlo mala vremenska ograničenja broj rešenih instanci veći od rešavača *najbolji-fiksirani* (na primer, za vremensko ograničenje od samo 1 sekunde, broj rešenih instanci se povećava sa 6216 na 6925, dok se ukupno vreme smanjuje sa 16.9 na 13.2 dana, gde se 1.4 dana upotrebljava za treniranje, a 11.8 za eksploataciju). Za približno isto vreme za koje je rešavač *najbolji-fiksirani* uspeo da reši 6216 instanci, naš pristup je rešio 7088 instanci (za vremensko ograničenje od 5 sekundi). Odnos između vremenskog ograničenja i ukupno utrošenog vremena je prikazan na slici 3.5. Slika pokazuje da ukupno vreme raste linearno sa povećanjem vremenskog ograničenja. Mislimo da u praktičnim scenarijima ima smisla koristiti vremensko ograničenje do 30 sekundi, u zavisnosti od dostupnog ukupnog vremena, pošto broj rešenih instanci vrlo sporo raste za veće vrednosti vremenskog ograničenja.

Može se postaviti još jedno važno pitanje: koliko gubimo na performansama korišćenjem predloženog pristupa u poređenju sa situacijom u kojoj već imamo portfolio



Slika 3.5: Poređenje utrošenog vremena portfolija ArgoCSP-kNN korišćenjem pristupa sa kratkim treniranjem i rešavača *najbolji-fiksirani*. Svaki znak na krivoj predstavlja jedan ishod zasnovan na rešavanju korpusa sa odgovarajućim vremenskim ograničenjem.

izgrađen nad instancama sličnih osobina instancama koje je potrebno rešiti? Dobra procena je već dostupna poređenjem sa rezultatima evaluacije predstavljenim u prethodnom poglavlju i datim u tabeli 3.6. Ako bi ArgoCSP-kNN bio treniran sa vremenskim ograničenjem od 600 sekundi, potrebno vreme za treniranje bi bilo 393.2 dana i bilo bi rešeno 7511 instanci. Sa druge strane, ako bi ArgoCSP-kNN bio korišćen sa pristupom kratkog treniranja sa vremenskim ograničenjem od 60 sekundi i vremenskim ograničenjem od 600 sekundi tokom eksploatacije, portfoliju bi bilo potrebno 53.8 dana za treniranje i rešio bi 7444 instanci (tabela 3.7). Razlika u 11 meseci potrebnih za treniranje je očigledno vrlo značajna, a razlika u broju rešenih instanci, možda nije toliko značajna – ostavljeno je korisniku da odluči, na osnovu svrhe i ukupnog dostupnog vremena.

3.6.2 Poređenje tehnika mašinskog učenja u pristupu kratkog treniranja

k-NN je jedna od najjednostavnijih tehnika mašinskog učenja. Zato je interesantno izvršiti poređenje sa nekim drugim tehnikama mašinskog učenja i videti da li bi trebalo zameniti *k-NN* nekom tehnikom mašinskog učenja koja bi dala bolje rezultate sa pristupom kratkog treniranja. Način na koji koristimo algoritam *k-NN* u portfoliju ArgoCSP-kNN je pomalo specifičan za problematiku kojom se bavimo,

i ne mogu svi algoritmi mašinskog učenja da se primene na isti način. Zato u eksperimentima koristimo malo drugačiji, ali potpuno prirodan dizajn portfolija (na primer, `SATzilla` koristi takav dizajn [134]), i takođe uključujemo ArgoCSP-kNN pošto je predstavljen u prethodnom poglavlju.

Dizajn portfolija korišćen u ovom poglavlju je zasnovan na predviđanju vremena izvršavanja (dakle, radi se o regresiji). U fazi eksploatacije se za svaki rešavač predviđa vreme izvršavanja na ulaznoj instanci pomoću prethodno naučenog prediktivnog modela za taj rešavač, i rešavač sa najmanjim predviđenim vremenom se pokreće da reši ulaznu instancu. U fazi treniranja modela, atributi svih trening instanci i PAR10 skorovi svakog od CSP rešavača na svim ovim instancama se koriste da se istrenira model za taj rešavač. Da bi se napravio model, više kombinacija vrednosti metaparametara se isprobava na trening skupu korišćenjem unakrsne provere sa 5 delova, a najbolja kombinacija se koristi za pravljenje prediktivnog modela.

Sproveli smo eksperimente sa regresionim tehnikama mašinskog učenja da bi procenili njihovu efikasnost. Eksperimentisali smo sa tehnikama: *k-NN*, linearnom regresijom i SVM. Sistem `RapidMiner`[4] je korišćen u eksperimentima. Kako se `RapidMiner` pokazao vrlo neefikasnim pri radu sa tehnikom SVM (proces se zaglavljivao sa mnogo različitih kombinacija metaparametara), uključili smo u evaluaciju i sistem `LIBSVM` [31] koji je pokazao veću stabilnost. Koristili smo dva tipa kernela: linearni i RBF.

Sa ciljem postizanja bolje efikasnosti, koristili smo podskup od 28 od mogućih 70 atributa (eliminisali smo polovinu atributa jer su imali vrednost 0 na svim instancama kao i nekoliko atributa koji su po našem mišljenju manje važni). Evaluaciju smo izvršili pomoću unakrsne provere sa 5 delova. Trening skup i test skup su *normalizovani* pre korišćenja – svaka od vrednosti atributa je skalirana na segment $[0, 1]$ deljenjem sa maksimalnom vrednošću tog atributa među instancama istog (trening ili test) skupa (svi atributi su nenegativni). Sve instance trening skupa su rešavane pomoću svakog od rešavača sa datim vremenskim ograničenjem, a potom su optimalni metaparametri izabrani na već opisan način. Metaparametri za *k-NN* su bili broj suseda k (kretao se u rasponu od 1 do 20) i funkcija udaljenosti d (4 funkcije udaljenosti već navedene u poglavlju 3.4). Metaparametri za SVM koji koristi linearni kernel bili su C (stepeni broja 2 od 2^{-6} do 2^{15}), ν (kreće se od 0.1 do 0.9 sa korakom 0.1). Za RBF kernel smo koristili iste kombinacije metaparametara kao za linearni kernel i dodatno metaparametar γ (stepeni broja 2 od

[4] `https://rapidminer.com/`

2^{-15} do 2^{5}). Metaparametri za linearnu regresiju bili su *ridge* (stepeni broja 10 od 10^{-8} do 10^{-1}), *use_bias* (true, false), *feature_selection* (none, M5 prime, greedy) i *eliminate_colinear_features* (true, false). Za svaku metodu, sve kombinacije metaparametara su isprobane na trening skupu i onda su one pomoću kojih je dobijen najbolji skor proglašene za optimalne.

Tabela 3.8 prikazuje dobijene rezultate. Tehnika *k-NN* implementira u okviru sistema `RapidMiner` je rešila značajno više instanci od ostalih tehnika, a ArgoCSP-kNN je rešio još više instanci (performanse ovog metoda sa smanjenim skupom atributa su skoro iste kao i sa kompletnim skupom, već predstavljenim u tabeli 3.7). Kako je cilj poređenje različitih tehnika mašinskog učenja korišćenjem kratkog treniranja, morali smo da ograničimo vreme upotrebljeno za predviđanje. Za oba kernela u `LIBSVM` testovima, meseci bi bili potrebni za završetak ovih eksperimenata, pa smo koristili smanjen skup vrednosti metaparametara koje su dale najbolje rezultate u preliminarnim eksperimentima. Čak i sa ovim smanjenim skupom parametara, `LIBSVM` testovima sa RBF kernelom je trebalo više od mesec dana da se završe. Zato zaključujemo da zbog previše utrošenog vremena, SVM modeli nisu pogodni za pristup kratkog treniranja.

Tabela 3.8: Rezultati eksperimentalne evaluacije korišćenjem različitih tehnika mašinskog učenja i različitih vremenskih ograničenja. U svim poljima je dat broj rešenih instanci.

	1	5	10	30	60
Linearna regresija (RM)	5947	5950	6073	6291	6459
SVM – linearni kernel (`LIBSVM`)	6216	6753	6753	6762	6695
SVM – RBF kernel (`LIBSVM`)	6235	6663	6518	6716	6748
k-NN (RM)	6052	6644	6804	7201	7309
k-NN (ArgoCSP-kNN)	6927	7095	7147	7375	7458

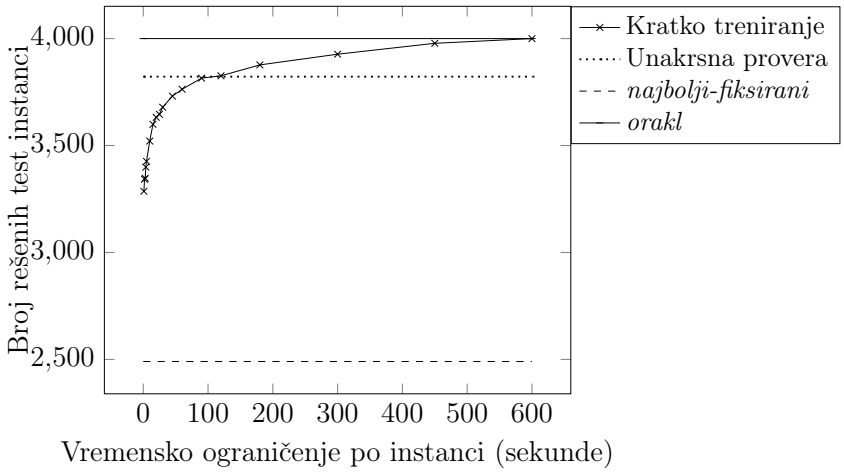
Kao i u poglavlju 3.5.3, procenjujemo performanse portfolija prethodno treniranih na sličnom (ali ne istom) skupu instanci, korišćenjem unakrsne provere sa 5 delova. Portfoliji koriste različite tehnike mašinskog učenja i vremensko ograničenje od 600 sekundi. Rezultati su dati u tabeli 3.9. U ovom slučaju, nije bilo smanjenja skupa metaparametara u slučaju SVM, ali su opet obe verzije algoritma *k-NN* efikasnije od ostalih pristupa.

Tabela 3.9: Rezultati evaluacije korišćenjem unakrsne provere sa 5 delova i različitih tehnika mašinskog učenja.

	Broj rešenih instanci
najbolji-fiksirani	6216
Linearna regresija (RM)	6515
SVM – linearni kernel (`LIBSVM`)	6715
SVM – RBF kernel (`LIBSVM`)	7323
k-NN (RM)	7432
k-NN (ArgoCSP-kNN)	7511
orakl	7631

3.6.3 Evaluacija na SAT instancama

Pošto je naša metodologija zasnovana na kratkom treniranju dala vrlo dobre rezultate na CSP portfoliju zasnovanom na pristupu ArgoSmArT k-NN, interesantno je pokazati da je ta metodologija takođe primenjiva i na SAT korpuse. Zato smo vršili eksperimente na SAT instancama korišćenjem originalnog ArgoSmArT k-NN portfolija.



Slika 3.6: Poređenje broja rešenih instanci portfolija ArgoSmArT k-NN na SAT instancama korišćenjem kratkog i standardnog treniranja, sa rešavačima *najbolji-fiksirani* i *orakl*. Svaki znak na krivoj predstavlja jedan ishod zasnovan na rešavanju korpusa sa odgovarajućim vremenskim ograničenjem.

Poredili smo ArgoSmArT k-NN sa i bez metodologije zasnovane na kratkom treniranju na instancama originalno korišćenim u radu koji je uveo ArgoSmArT k-NN [102] – instance sa takmičenja SAT Competitions (2002-2007) i SAT Races

(2000-2008). U pristupu kratkog treniranja, portfolio se i trenira i pokreće na tim instancama. U originalnom pristupu bez kratkog treniranja, portfolio je evaluiran korišćenjem unakrsne provere sa 5 delova. Rezultati poređenja su dati na slici 3.6. Pošto u originalnom pristupu ne postoji niz vremenskih ograničenja, taj pristup je predstavljen linijom koja je paralelna horizontalnoj osi. Kriva za pristup koji koristi kratko treniranje izgleda vrlo slično onoj koja je predstavljena u prethodnom poglavlju i vrlo dobri rezultati su dobijeni za vrlo mala vremenska ograničenja. Zato, naša metodologija može efikasno biti primenjena ne samo na CSP instance, nego i na SAT instance.

Glava 4

Primene CSP rešavača

U ovoj glavi ćemo predstaviti dve primene CSP rešavača (korišćen je sistem *meSAT* ali i drugi sistemi). Prva primena se odnosi na rešavanje problema raspoređivanja kontrolora letetnja, a druga na generisanje i rešavanje velikih sudoku zagonetki.

4.1 Problem raspoređivanja kontrolora letenja i njegovo rešavanje

Problemi raspoređivanja zaposlenih su intenzivno proučavani u poslednjih nekoliko decenija (na primer, problem raspoređivanja medicinskih sestara [27], problem pravljenja rasporeda časova [33], itd.). Ulazni parametri ovih problema (na primer, broj raspoloživih radnika, njihove veštine i veštine neophodne za različite pozicije) i ograničenja (na primer, maksimalan broj uzastopnih radnih dana za svakog radnika) se prvo moraju odrediti i na osnovu ulaznih parametara, potrebno je napraviti raspored koji zadovoljava navedena ograničenja.

U ovom poglavlju razmatramo problem koji se zove *Problem raspoređivanja kontrolora leta* (eng. *Air Traffic Controller (ATCo) Shift Scheduling Problem*), skraćeno *ATCoSSP*. Cilj je napraviti raspored smena, tako da je u svakom radnom satu svaka od pozicija popunjena dovoljnim brojem kontrolora sa neophodnim veštinama. Zbog prirode i značaja svog posla, kontrolori moraju biti potpuno koncentrisani dok su na poziciji. Zato njihov raspored mora da zadovoljava znatno striktniji skup ograničenja u poređenju sa drugim problemima raspoređivanja zaposlenih i svako nezadovoljeno ograničenje može predstavljati pretnju za ljudske živote i bezbednost. Pre

objavljivanja rada autora ove teze [120] nisu postojali radovi na temu opisa modela ovog problema kao ni detaljnog opisa metoda rešavanja (koliko je autoru ove teze poznato). Tekst koji sledi je najvećim delom zasnovan na pomenutom radu.

4.1.1 Pregled postojećih primena

Istraživanja vezana za *ATCoSSP*. Pregled dostupnih rezultata koji se odnose na *ATCoSSP* su napravili Arnvig sa koautorima [6]. U tom dokumentu je predstavljen pregled literature koja se bavi uticajem rada u smenama na zdravlje, bezbednost, efikasnost i socijalni aspekt života zaposlenih, sa naročitim osvrtom na zaposlene uključene u vazdušnom saobraćaju. Predstavljeni su i neki pravci vezani za dalje poboljšanje rada u smenama. Dokument koji je izradio EUROCONTROL, Evropska Organizacija za bezbednost u vazdušnom saobraćaju ([48]), bavi se utvrđivanjem dobrih praksi pri pravljenju rasporeda smena, u avionskom transportu ali i van njega (na primer, u bolnicama, policiji). Ciljevi tog tokumenta su pravljenje pregleda uobičajenih rešenja smenskog rada kao i povećanje bezbednosti i poboljšanje zdravlja zaposlenih.

Neke softverske alatke za generisanje rasporeda kontrolora postoje [48], a uočene su i prednosti i mane korišćenja ovih alatki [129]. Neke od njih su alatke razvijene u okviru kompanija i više podataka o njima nije dostupno. Druge su opštije alatke (na primer, *Shift Scheduler Continuous*[1]) i mogu biti korišćeni samo sa ograničenim verzijama problema *ATCoSSP* (na primer, kontrolori se dele u timove i ovo je obično slučaj kod velikih aerodroma). Kako je navedeno u literaturi (poglavlje 6.3 dokumenta [48]), većina korišćenih softverskih alatki ne uzima u dovoljnoj meri u obzir potrebu za određenim brojem kontrolora u zavisnosti od količine posla.

Slični problemi i metode za njihovo rešavanje. Problemi raspoređivanja su intenzivno proučavani u literaturi, pa je jedan pravac rešavanja problema *ATCoSSP* identifikacija sličnih problema i njihovih metoda rešavanja, i prilagođavanje tih metoda raspoređivanju kontrolora. Jedan od veoma poznatih i proučavanih problema je *Problem raspoređivanja medicinskih sestara* (eng. *Nurse Scheduling Problem*), skraćeno NSP [27]. Iako postoje mnoge sličnosti između ovog problema i problema koji mi razmatramo, važna razlika je da (u svojoj uobičajenoj formi) NSP ne uključuje raspoređivanje na nivou sata. *Pravljenje rasporeda časova* (eng. *Course Timetabling*), skraćeno CTT [33], je dobro proučavan problem kod koga je potrebno napra-

[1] `http://www.bizpeponline.com/Helpssce.html`

viti raspored na nivou sata. *Raspoređivanje studenata po ispitima* je takođe jedan dosta proučavan problem [29]. *ATCoSSP* se razlikuje od navedenih problema jer su zahtevi pri raspoređivanju mnogo striktniji. Striktniji zahtevi povlače veliki broj *tvrdih ograničenja* (eng. *hard constraints*) koja moraju biti zadovoljena, pa je nalaženje ispravnog rasporeda kontrolora često vrlo težak zadatak. Najefikasnije metode u rešavaju problema NSP i CTT su metaheuristike i pri njihovom korišćenju se pretpostavlja da se početno rešenje može naći na neki jednostavan način. Opisane razlike utiču da prilagođavanje poznatih metaheuristika problemu *ATCoSSP* nije jednostavno i lako primenjivo, kao u slučaju problema sa sličnim karakteristikama kao NSP i CTT.

4.1.2 Opis problema raspoređivanja kontrolora leta

ATCoSSP je problem dodeljivanja smena kontrolorima u periodu koji se razmatra (obično je period mesec ili godina) uz ispunjenje određenih uslova. Postoje mnogi dokumenti koji opisuju ove uslove ([6], [48], [51]). Opis problema koji sledi ni na koji način nije sveobuhvatan, jer rasporedi imaju različite ulazne parametre i moraju zadovoljavati ograničenja koja se mogu značajno razlikovati od aerodroma do aerodroma. U nastavku sledi sažet opis problema *ATCoSSP*, a u narednom poglavlju je predstavljen formalniji i detaljniji opis ovog problema, uvođenjem tri njegova modela.

Raspored se bavi za određeni period, period se sastoji iz određenog broja dana, a dani se sastoje iz fiksiranog broja *termina*. Svakog dana kontroloru je dodeljena tačno jedna od tri mogućih *aktivnosti*. Tokom *smena* (eng. *working shifts*) kontrolor radi na aerodromu datog dana od prvog do poslednjeg termina te smene (uključujući oba ta termina), a relaksira se tokom ostalih termina tog dana. Smene mogu biti različitih dužina i u zavisnosti od prvog termina, razlikujemo jutarnje, popodnevne i noćne smene. Pretpostavlja se da su termini smena poznati unapred. Ako kontrolor ne radi smenu određenog dana, kažemo da kontrolor tog dana ima *dan relaksacije* (eng. *rest day*) (odgovara danima vikenda kod većine zanimanja). Svaki kontrolor ima na raspolaganju određen broj plaćenih *dana odmora* (eng. *vacation days*). Za razliku od dana relaksacije kojih je obično 1-2 u nizu i koji su neophodni zbog izbegavanja prevelikog opterećenja i zadržavanja produktivnosti kontrolora, dani odmora se obično uzimaju tako da ih je više u nizu (na primer, 7 ili 10). Za svaki

period se traženi odmori od strane kontrolora odobravaju ili ne odobravaju unapred od strane uprave.

Za svakog kontrolora, broj *radnih sati* (eng. *working hours*) u razmatranom periodu mora biti veći od *min* sati (da bi kontrolor dobio punu platu) i manji od *max* sati (da bi se izbeglo preopterećenje). Svaka smena povlači broj radnih sati jednak trajanju te smene. Dan relaksacije se ne računa u radne sate. *Za svaki dan odmora, kontroloru se uračunava neki propisima utvrđen broj radnih sati*, pa ćemo u okviru modela koristiti da se tokom određenih fiksiranih sati (na primer, od 8 do 16h) kontroloru računaju radni sati u okviru odmora, a da tokom ostalih sati kontrolor ima sate relaksacije. Računanje dana odmora u radne sate omogućava da kontrolori i tokom peroda kada uzimaju odmor dobijaju punu platu.

Nijedan kontrolor ne sme da radi više od određenog broja uzastopnih smena ili da ima više od određenog broja uzastopnih dana relaksacije (ove dve vrednosti su obično 2 ili 3). Samo pojedini kontrolori imaju licencu da budu šefovi smena. Svakog dana i u svakom terminu kada aerodrom radi, barem jedan od šefova smena mora biti na aerodromu[2]. Svaki kontrolor mora imati određeni minimalan broj dana relaksacije svakog meseca (ukoliko kontrolor uzima dane odmora u tom mesecu, ovaj broj se računa po formuli određenoj propisima). Kontrolorima je potrebna relaksacija između smena i propisi određuju minimalan broj termina relaksacije između dve smene (na primer, broj termina koji je u zbiru jednak 12 sati).

Dodela pozicija kontrolorima u okviru smena koje rade je takođe deo problema *ATCoSSP*. Postoje različite vrste pozicija na aerodromima (na primer, toranj, terminal, oblasna) i u zavisnosti od veličine aerodroma neke ili sve pozicije postoje. U svakom terminu smene kontrolor može biti ili na poziciji ili može imati termin relaksacije. U svakom terminu kontroloru može biti dodeljena maksimum jedna pozicija. U dva uzastopna termina kontrolor može biti na dve različite pozicije. Kontrolor ne sme biti na poziciji više od datog broja uzastopnih termina. Na osnovu očekivanog intenziteta vazdušnog saobraćaja, svakog dana i u svakom terminu kada aerodrom radi (neki aerodromi rade 24 sata dok drugi ne rade) određeni broj kontrolora je potreban za svaku poziciju. Kontrolor mora da poseduje specifične veštine da bi dobio licencu za rad na određenoj poziciji. Podrazumeva se da su licence kontrolora i broj potrebnih kontrolora za svaki termin informacije koje su na raspolaganju pre pravljenja rasporeda[3].

[2]Ovaj zahtev može da bude izražen i na drugi način, korišćenjem smena umesto termina.

[3]U praksi je određivanje broja potrebnih kontrolora problem za sebe kome je posvećena posebna pažnja u literaturi ([51], [129]).

Opis se do sada fokusirao samo na tvrdim ograničenjima koja su neophodna za ispravnost rasporeda i zato moraju biti zadovoljena. *Meka ograničenja* (eng. *soft constraints*) predstavljaju želje (ili preference) osoblja. Kontrolori mogu preferirati različite smene (na primer, mogu preferirati jutarnje smene), mogu želeti da rade smene uzastopnim danima što je ređe moguće, itd. Razlozi za uključivanje osbolja u pravljenje rasporeda kao i neke uobičajene preference su opisane od strane Arnviga sa koautorima [6] (poglavlje 6.5).

4.1.3 Modeli problema raspoređivanja kontrolora leta

Pre nego što uvedemo modele problema, navešćemo neke pretpostavke koje smo napravili. Iako termini mogu biti bilo koje fiksirane dužine, u ovoj tezi pretpostavljamo da je dužina termina 1 sat, pa ćemo i ograničenja zadavati na nivou sata. Rasporedi smena se generišu za period od jednog meseca i za svaki mesec, novi raspored smena mora biti generisan. Postoje mnogi razlozi za ovo: očekivani mesečni intenzitet saobraćaja se menja, kontrolori koriste dane odmora u različitim mesecima, itd. Pretpostavljamo da postoje dve vrste pozicija: toranj i terminal, kao i da ne postoje noćne smene. Navedene pretpostavke nisu ograničavajuće, pošto se modeli mogu lako proširiti da podrže više tipova pozicija i noćne smene.

Pretpostavimo da su dani $1, \ldots, n_d$, kontrolori $1, \ldots, n_c$, a moguće aktivnosti za bilo koji dan $1, \ldots, n_s$[4]. Da bi napravili kompaktnije i efikasnije modele, pretpostavljamo da su aktivnosti podeljene tako da su radne smene $1, \ldots, n_s - 2$, da je $rel = n_s - 1$ dan relaksacije, a $vac = n_s$ dan odmora. Sati imaju vrednosti $0, \ldots, 23$, a za svaku smenu ili dan odmora s, prvi (s_f) i poslednji (s_l) radni sat su fiksirani (već je rečeno da se tokom dana odmora određeni sati računaju kao radni, iako kontrolor ne radi tokom odmora).

Eksperimentisali smo sa raznim modelima i ograničenjima. U naredna 3 poglavlja, opisana su 3 modela koja su se pokazala efikasnim u praksi, a nakon toga opisane su metode rešavanja optimizacionih instanci. Prva dva modela su razvijena da budu pogodna za CSP rešavače, dok je treći razvijen da bude pogodan za rešavače problema SAT i njemu bliskih problema. Sva 3 modela su u stanju lučne konzistencije (termin opisan u poglavlju 2.2.2.2), ali ostale vrste lokalne konzistencije nisu garantovane (na primer, čvorna konzistencija). U CP zajednici se lučna konzistencija pokazala najbitnijom vrstom lokalne konzistencije u praksi, pa je ovo

[4]Skraćeniće potiču od oznaka na engleskom, na primer, d od *day*, c od *controller*, s od *shift*. Ovo važi za sve skraćeniće koje se javljaju u ovom poglavlju.

razlog zašto smo za cilj imali postizanje baš ove vrste konzistencije. Obezbeđivanjem lučne konzistencije postignuto je da rešavači ne gube vreme na početku rešavanja na eventualno dovođenje problema u stanje ove konzistencije.

Samo vrednosti promenljivih koje određuju smene kontrolora za svaki dan i pozicije kontrolora za svaki sat se koriste kada se pravi tabelarni raspored za zaposlene. Ostale promenljive su pomoćne i koriste se da unaprede čitljivost i efikasnost modela. Činjenica da su dani odmora fiksirani za period omogućava da se modeli učine kompaktnijima.

4.1.3.1 CSP model

Ovaj model koristi ograničenja linearne aritmetike i globalno ograničenje *count* (ono je opisano u poglavlju 2.1.2). U okviru ovog modela smatramo da se kontroloru računaju radni sati tokom fiksiranih sati dana kada uzima odmor, a sati relaksacije tokom ostalih sati tih dana.

Celobrojne promenljive.

- $dc_{d,c}$: na dan d kontroloru c[5] (indeksi se tokom rešavanja problema zamenjuju konkretnim vrednostima) može biti dodeljena bilo koja od aktivnosti $1, \ldots, n_s$ (neka od smena, dan relaksacije ili dan odmora). Primetimo da činjenica da su vrednosti smena $1, \ldots, n_s - 2$ omogućava da naznačimo da konrolor c radi (neku) smenu na dan d uvođenjem ograničenja $dc_{d,c} \leq n_s - 2$.

- $dhc_{d,h,c}$: u satu h na dan d kontroloru c mogu biti dodeljeni različiti zadaci: c može biti na poziciji na tornju ($TOW = 0$) ili terminalu ($TER = 1$), c može imati sat relaksacije na aerodromu ($B = 2$), ili sat odmora koji se računa kao radni sat ($V = 3$) ili sat za relaksaciju van posla ($R = 4$) (bilo u okviru dana relaksacije ili dana odmora). Primetimo da nam uvedene vrednosti omogućavaju da naznačimo da c radi na aerodromu tokom sata h na dan d uvođenjem ograničenja $dhc_{d,h,c} \leq B$.

- $h_{d,c}$: na dan d kontroloru c se računa određeni broj radnih sati $(0, \ldots, 12)$.

[5]Većina ograničenja mora da bude tačna za sve kontrolore, ali u opisima koristimo nekog fiksiranog konrolora c. Slično važi za dane, sate i smene.

Veze između promenljivih.

- Ako kontrolor c radi smenu s na dan d, onda c radi tokom radnih sati te smene i ima sate relaksacije tokom ostalih sati u toku tog dana. Zato, za svako $j \in \{s_f, \ldots, s_l\}$ važi: $dc_{d,c} = s \rightarrow dhc_{d,j,c} \leq B$. Za svako $j \in \{0, \ldots, 23\} \setminus \{s_f, \ldots, s_l\}$ važi: $dc_{d,c} = s \rightarrow dhc_{d,j,c} = R$.

- Ako kontrolor c ima dan relaksacije na dan d, onda c ima sate relaksacije tokom svih sati tog dana. Dakle, za svako $j \in \{0, \ldots, 23\}$: $dc_{d,c} = rel \rightarrow dhc_{d,j,c} = R$.

- Ako kontrolor c radi na aerodromu tokom sata h na dan d onda c radi neku smenu tog dana: $dhc_{d,h,c} \leq B \rightarrow dc_{d,c} \leq n_s - 2$.

- Ako kontrolor c na dan d ima aktivnost s sa radnim satima $s_f, \ldots, s_l$, pri čemu je s ili smena ili dan odmora, onda c radi $s_l - s_f + 1$ sati tog dana: $dc_{d,c} = s \rightarrow h_{d,c} = s_l - s_f + 1$. U slučaju dana relaksacije uvodi se ograničenje: $dc_{d,c} = rel \rightarrow h_{d,c} = 0$.

Tvrda ograničenja.

- *Dodela smena:* Na dan d kontroloru c se dodeljuje jedna od mogućih aktivnosti (ovo ograničenje nije potrebno eksplicitno uvoditi pošto promenljiva $dc_{d,c}$ uzima tačno jednu od vrednosti $1, \ldots, n_s$).

- *Dani odmora:* Ako je kontroloru c odobren dan odmora vac na dan d, onda je $dc_{d,c} = vac$. Pritom se određeni sati računaju kao radni sati, a ostali sati dana kao sati relaksacije. Dakle, za bilo koje $j \in \{vac_f, \ldots, vac_l\}$ važi: $dhc_{d,j,c} = V$. Za bilo koje $j \in \{0, \ldots, 23\} \setminus \{vac_f, \ldots, vac_l\}$ važi: $dhc_{d,j,c} = R$. Ako kontrolor c ne uzima odmor na dan d, onda je $dc_{d,c} \neq vac$ i za svako $j \in \{0, \ldots, 23\}$ važi: $dhc_{d,j,c} \neq V$.

- *Uzastopne smene:* Kontrolor c mora imati barem jedan dan relaksacije tokom cws (cws je skraćenica engleskog termina "consecutive working shifts") dana za redom počev od dana d, kad god c ne uzima dan odmora nekog od ovih dana: $count(\{dc_{d,c}, \ldots, dc_{d+cws-1,c}\},\ rel) \geq 1$.

- *Uzastopni dani relaksacije:* Kontrolor c ne sme da ima više od crs (crs je skraćenica engleskog termina "consecutive rest shifts") dana relaksacije za redom počev od bilo kog dana d: $count(\{dc_{d,c}, \ldots, dc_{d+crs,c}\},\ rel) \leq crs$ (od bilo kojih $crs + 1$ dana, najviše crs je dana relaksacije).

- *Šefovi smena:* Na dan d i radni sat aerodroma h jedan od kontrolora koji ima licencu da bude šef smene mora raditi na aerodromu (može imati i sat relaksacije). Neka su $c_1, \ldots, c_p$ kontrolori sa takvom licencom koji nisu na odmoru u nekom satu h. Barem jedan od njih ne sme imati sat relaksacije tokom sata h:
  $count(\{dhc_{d,h,c_1}, \ldots, dhc_{d,h,c_p}\},\ R) \neq p$.

- *Minimalan broj dana relaksacije:* Kontrolor c mora imati minimalno m dana relaksacije tokom meseca: $count(\{dc_{1,c}, \ldots, dc_{n_d,c}\},\ rel) \geq m$. Vrednost m se računa pomoću formule koja kao parametar ima broj uzetih dana odmora.

- *Vreme relaksacije između smena:* Ako kontrolor c radi smenu s_i na dan d, onda postoje određene smene koje kontrolor c ne sme raditi na dan $d+1$, jer mu je potreban određeni broj sati relaksacije između dve smene. Za svaku takvu smenu s_j važi: $dc_{d,c} = s_i \rightarrow dc_{d+1,c} \neq s_j$.

- *Maksimalan broj radnih sati u 2 dana:* U sumi na dan d i narednog dana kontrolor c ne sme raditi više nego propisima određen broj q radnih sati (na primer, $q = 22$): $h_{d,c} + h_{d+1,c} \leq q$.

- *Maksimalan broj radnih sati:* Kontrolor c ne sme da radi više nego propisima određen broj max radnih sati tokom meseca: $\sum_{d=1}^{n_d} h_{d,c} \leq max$.

- *Minimalan broj radnih sati:* Kontrolor c mora da radi minimalno min broj radnih sati tokom meseca da bi dobio punu platu: $\sum_{d=1}^{n_d} h_{d,c} \geq min$.

- *Jedna lokacija po satu:* Kontrolor c u satu h na dan d može da bude tačno na jednom mestu: c može biti ili na poziciji na tornju ili na poziciji na terminalu ili može da ima sat relaksacije na aerodromu ili mu se računa radni sat tokom dana odmora ili da ima sat relaksacije. Za ovo nije potrebno uvesti ograničenja pošto promenljiva $dhc_{d,h,c}$ može uzeti tačno jednu od vrednosti $0, \ldots, 4$.

- *Pozicije popunjene:* Ako je u satu h na dan d najmanje k kontrolora potrebno za poziciju na tornju, mora važiti: $count(\{dhc_{d,h,1}, \ldots, dhc_{d,h,n_c}\},\ TOW) \geq k$. Analogno važi za poziciju na terminalu[6].

- *Uzastopan broj sati na poziciji:* Na dan d počev od sata h kontrolor c ne sme biti na poziciji više od m sati za redom: $dhc_{d,h,c} \geq B \vee \ldots \vee dhc_{d,h+m,c} \geq B$.

[6]Više ograničenja *count* u praksi zamenjujemo jednim ograničenjem *global cardinality* [12] da bi se dobio efikasniji model, ali izbegavamo navođenje ovakvih detalja.

- *Licence:* Ako kontrolor c nema licencu da bude na poziciji na tornju, onda za svaki dan d i sat h mora da važi: $dhc_{d,h,c} \neq TOW$. Analogno važi za poziciju na terminalu.

Meka ograničenja. Želje su izražene na nivou dana – na primer, ako kontrolor preferira neku od smena svakog dana, ovo se prevodi u broj želja jednak broju dana (za svaki dan po jedna želja). Svaka želja kontrolora je predstavljena pomoću ograničenja koje je zadovoljeno akko želja nije ispunjena. Svako od ograničenja je postavljeno da bude ekvivalentno novoj celobrojnoj promenljivoj čiji je domen $\{0, 1\}$ – ograničenje je zadovoljeno akko odgovarajuća promenljiva ima vrednost 0. Ako sve ove promenljive uzimaju vrednost 0, onda su sve želje ispunjene. Označamo svaku novu promenljivu sa $x_{c,i}$, gde za svakog fiksiranog kontrolora c indeks i uzima broj vrednosti jednak broju mekih ograničenja uvedenih za tog kontrolora. Meka ograničenja koja koristimo su:

- *Preference smena:* Ako kontrolor c preferira smene $s_1, \ldots, s_z$, onda je svaka smena s različita od ovih smena nepoželjna bilo kog dana d: $x_{c,i} \leftrightarrow dc_{d,c} = s$. Na primer, ako mesec ima 30 dana i ako je najmanji neiskorišćen nenegativan indeks promenljivih povezanih sa kontrolorom c indeks j, onda se uvodi 30 promenljivih $x_{c,j}, \ldots, x_{c,j+29}$.

- *Minimizacija uzastopnih smena:* Kontrolor c preferira da radi smene uzastopnim danima što je ređe moguće. Za svaki dan d važi:
  $x_{c,i} \leftrightarrow dc_{d,c} \leq n_s - 2 \wedge dc_{d+1,c} \leq n_s - 2$.

- *Maksimizacija uzastopnih dana relaksacije:* Kontrolor c ne preferira izolovane dane relaksacije. Za svaka 3 uzastopna dana $d, d+1, d+2$, takva da c ne uzima dan odmora tih dana, važi:
  $x_{c,i} \leftrightarrow dc_{d,c} \leq n_s - 2 \wedge dc_{d+1,c} = rel \wedge dc_{d+2,c} \leq n_s - 2$.

4.1.3.2 Linearni model

Kako sintaksa nekih rešavača ne dozvoljava korišćenje globalnih ograničenja, prilagodili smo CSP model da ne koristi ova ograničenja. Celobrojne promenljive, veze između promenljivih i meka ograničenja su ista kao kod CSP modela. Zato ovde jedino predstavljamo tvrda ograničenja.

Tvrda ograničenja. Većina tvrdih ograničenja su ista kao kod CSP modela, pa opisujemo samo ograničenja koja se modeliraju na drugačiji način. Za `if`-then-else izraze se uvode nove promenljive i ograničenja koja opisuju veze tih promenljivih, pa ovaj model može biti znatno glomazniji od CSP modela.

- *Uzastopne smene:* Kontrolor c mora imati barem jedan dan relaksacije tokom cws dana za redom počev od dana d, kad god c ne uzima dan odmora nekog od tih dana: $dc_{d,c} = rel \vee \ldots \vee dc_{d+cws-1,c} = rel$.
- *Uzastopni dani relaksacije:* Kontrolor c ne sme da ima više od crs dana relaksacije za redom počev od bilo kog dana d: $dc_{d,c} \neq rel \vee \ldots \vee dc_{d+crs,c} \neq rel$.
- *Šefovi smena:* Na dan d i radni sat aerodroma h, jedan od kontrolora $c_1, \ldots, c_p$ koji ima licencu da bude šef smene mora raditi na aerodromu: $dhc_{d,h,c_1} \leq B \vee \ldots \vee dhc_{d,h,c_p} \leq B$.
- *Minimalan broj dana relaksacije:* Kontrolor c mora imati minimalno m dana relaksacije tokom meseca: $\sum_{d=1}^{n_d}(\texttt{if } (dc_{d,c} = rel) \texttt{ then } 1 \texttt{ else } 0) \geq m$.
- *Pozicije popunjene:* U satu h na dan d najmanje k kontrolora potrebno za poziciju na tornju: $\sum_{c=1}^{n_c}(\texttt{if } (dhc_{d,h,c} = TOW) \texttt{ then } 1 \texttt{ else } 0) \geq k$. Analogno važi za poziciju na terminalu.

4.1.3.3 Bulovski model

U našem predstavljanju ograničenja bulovskog modela koristimo ekvivalencije, implikacije i klauze što je češće moguće da bi unapredili jasnoću teksta, ali ovaj model koristi samo bulovska ograničenja kardinalnosti, opisana u poglavlju 2.3.2. Svaka ekvivalencija se može prevesti u 2 implikacije. Implikacija $a \rightarrow b$ se može direktno prevesti u klauzu $\neg a \vee b$, a komplikovanije implikacije se mogu prevesti u klauze korišćenjem De Morganovih zakona i pravila distributivnosti[7]. Primetimo da je svaka klauza $l_1 \vee \ldots \vee l_n$ isto što i bulovsko ograničenje kardinalnosti $l_1 + \ldots + l_n \geq 1$.

Iskazne promenljive.

- $dcs_{d,c,s}$: na dan d kontrolor c ima aktivnost s.

[7]Ne postoji rizik od eksponencijalnog uvećanja pošto implikacije u ovom modelu imaju mali broj literala.

- $dc_{d,c}$: na dan d kontrolor c radi neku smenu.
- $dhc_{d,h,c}$: sat h na dan d za kontrolora c je radni sat (sat na aerodromu ili sat na odmoru koji mu se računa kao radni sat). Ako je ova promenljiva netačna, c ima sat relaksacije.
- $tow_{d,h,c}/ter_{d,h,c}/pos_{d,h,c}$: tokom sata h na dan d kontrolor c je na poziciji na tornju/poziciji na terminalu/bilo kojoj poziciji.
- $b_{d,h,c}/v_{d,h,c}$: tokom sata h na dan d kontrolor c ima sat relaksacije na aerodromu/sat na odmoru koji mu se računa kao radni sat.

Veze između promenljivih.

- Kontroloru c se računa radni sat tokom sata h na dan d akko je c na poziciji na tornju ili na poziciji na terminalu ili ima sat relaksacije na aerodromu ili ima sat odmora koji se računa kao radni sat: $dhc_{d,h,c} \leftrightarrow tow_{d,h,c} \vee ter_{d,h,c} \vee b_{d,h,c} \vee v_{d,h,c}$.
- Ako kontrolor c radi smenu s na dan d, onda c radi tokom radnih sati te smene i ima sate relaksacije tokom ostatka toga dana. Dakle, za bilo koje $j \in \{s_f, \ldots, s_l\}$ važi: $dcs_{d,c,s} \rightarrow dhc_{d,j,c}$. Za bilo koje $j \in \{0, \ldots, 23\} \setminus \{s_f, \ldots, s_l\}$ važi: $dcs_{d,c,s} \rightarrow \neg dhc_{d,j,c}$.
- Ako kontrolor c ima dan relaksacije na dan d, onda c ne radi (ima sate relaksacije) tokom svih sati tog dana. Dakle, za bilo koje $j \in \{0, \ldots, 23\}$ važi: $dcs_{d,c,rel} \rightarrow \neg dhc_{d,j,c}$.
- Kontrolor c je na poziciji tokom sata h na dan d akko je c na poziciji na tornju ili na terminalu: $pos_{d,h,c} \leftrightarrow tow_{d,h,c} \vee ter_{d,h,c}$.
- Kontrolor c radi smenu na dan d akko c tog dana nema ni dan relaksacije ni dan odmora: $dc_{d,c} \leftrightarrow \neg dcs_{d,c,vac} \wedge \neg dcs_{d,c,rel}$,

Tvrda ograničenja.

- *Dodela smena:* Na dan d kontrolor c radi jednu od smena, ima dan relaksacije ili dan odmora: $dcs_{d,c,1} + \ldots + dcs_{d,c,n_s} = 1$.

- *Dani odmora:* Ako kontrolor c uzima dan odmora na dan d, onda važi da je $dcs_{d,c,vac} = 1$. Ovo povlači i određeni broj sati odmora koji se računaju kao radni i vreme relaksacije tokom ostalih sati tog dana. Dakle, za bilo koje $j \in \{vac_f, \ldots, vac_l\}$ važi: $v_{d,j,c} = 1$. Za bilo koje $j \in \{0, \ldots, 23\} \setminus \{vac_f, \ldots, vac_l\}$ važi: $dhc_{d,j,c} = 0$. Ako kontrolor c ne uzima dan odmora na dan d, onda je $dcs_{d,c,vac} = 0$ i za bilo koje $j \in \{0, \ldots, 23\}$ važi: $v_{d,j,c} = 0$.

- *Uzastopne smene:* Kontrolor c mora imati barem jedan dan relaksacije tokom cws dana za redom počev od nekog dana d, kadgod c ne uzima dan odmora nekog od tih dana: $\neg dc_{d,c} \vee \ldots \vee \neg dc_{d+cws-1,c}$.

- *Uzastopni dani relaksacije:* Kontrolor c ne sme da ima više od crs dana relaksacije za redom počev od bilo kog dana d: $\neg dcs_{d,c,rel} \vee \ldots \vee \neg dcs_{d+crs,c,rel}$.

- *Šefovi smena:* Na dan d i radni sat aerodroma h jedan od kontrolora koji ima licencu da bude šef smene mora biti na aerodromu. Neka su $c_1, \ldots, c_p$ kontrolori sa takvom licencom koji nemaju sat odmora u satu h. Barem jedan od njih mora raditi na aerodromu:
$tow_{d,h,c_1} \vee ter_{d,h,c_1} \vee b_{d,h,c_1} \vee \ldots \vee tow_{d,h,c_p} \vee ter_{d,h,c_p} \vee b_{d,h,c_p}$.

- *Minimalan broj dana relaksacije:* Kontrolor c mora imati minimalno m dana relaksacije tokom meseca: $dcs_{1,c,rel} + \ldots + dcs_{n_d,c,rel} \geq m$.

- *Vreme relaksacije između smena:* Ako kontrolor c radi smenu s_i na dan d, onda postoje neke smene koje kontrolor c ne sme raditi na dan $d + 1$, jer je kontroloru c potreban određeni broj sati relaksacije. Za svaku takvu smenu s_j uvodi se sledeće ograničenje: $dcs_{d,c,s_i} \rightarrow \neg dcs_{d+1,c,s_j}$.

- *Maksimalan broj radnih sati:* Kontrolor c ne sme da radi više od propisima određenog broja max radnih sati tokom meseca: $\sum_{d=1}^{n_d} \sum_{h=0}^{23} dhc_{d,h,c} \leq max$.

- *Minimalan broj radnih sati:* Kontrolor c mora da radi minimalno min broj radnih sati tokom meseca da bi dobio punu platu: $\sum_{d=1}^{n_d} \sum_{h=0}^{23} dhc_{d,h,c} \geq min$.

- *Jedna lokacija po satu:* Kontrolor c u satu h na dan d može da bude najviše na jednom mestu: c može biti ili na poziciji na tornju ili na poziciji na terminalu ili može da ima sat relaksacije na aerodromu ili mu se računa radni sat tokom dana odmora: $tow_{d,h,c} + ter_{d,h,c} + b_{d,h,c} + v_{d,h,c} \leq 1$ (ne stoji znak stroge jednakosti pošto kontrolor može imati i sat relaksacije).

- *Pozicije popunjene:* Ako je u satu h na dan d najmanje k kontrolora potrebno za poziciju na tornju, uvodi se sledeće ograničenje: $\sum_{c=1}^{n_c} tow_{d,h,c} \geq k$. Analogno važi za poziciju na terminalu.

- *Uzastopan broj sati na poziciji:* Na dan d počev od sata h kontrolor c ne sme biti na poziciji više od m sati za redom: $\neg pos_{d,h,c} \vee \ldots \vee \neg pos_{d,h+m,c}$.

- *Licence:* Ako kontrolor c nema licencu da bude na poziciji na tornju, onda za svaki dan d i sat h mora da važi: $\neg tow_{d,h,c}$. Analogno važi za poziciju na terminalu.

Meka ograničenja. Nove iskazne promenljive se uvode po istom principu kao celobrojne promenljive sa domenom $\{0, 1\}$ kod CSP modela.

- *Preference smena:* Ako kontrolor c preferira smene $s_1, \ldots, s_z$, onda je svaka smena s različita od ovih smena nepoželjna bilo kog dana d: $x_{c,i} \leftrightarrow dcs_{d,c,s} = 1$.

- *Minimizacija uzastopnih smena:* Kontrolor c preferira da radi smene uzastopnim danima što je ređe moguće. Za svaki dan d važi: $x_{c,i} \leftrightarrow dc_{d,c} \wedge dc_{d+1,c}$.

- *Maksimizacija uzastopnih dana relaksacije:* Kontrolor c ne preferira izolovane dane relaksacije. Za svaka 3 uzastopna dana $d, d+1, d+2$, takve da c ne uzima dan odmora tih dana, važi: $x_{c,i} \leftrightarrow dc_{d,c} \wedge dcs_{d+1,c,rel} \wedge dc_{d+2,c}$.

4.1.4 Tehnike traženja optimalne vrednosti

Kontrolori određuju značaj njihovih želja i ovo se izražava pridruživanjem celobrojne težine svakoj želji koju imaju. Da bi se raspored učinio pravednijim, težine se skaliraju tako da je suma težina svakog kontrolora jednaka nekoj fiksiranoj vrednosti. Ako kontrolor c ima m_c želja predstavljenih pomoću iskaznih promenljivih $x_{c,i},\ i \in \{1, \ldots, m_c\}$ (uvedenih u opisu mekih ograničenja) i ako su pridružene težine skalirane na vrednosti $w_{c,i},\ i \in \{1, \ldots, m_c\}$, onda je *penal kontrolora* definisan pomoću: $\texttt{penal}_c = \sum_{i=1}^{m_c} w_{c,i} \cdot x_{c,i}$. Za svakog kontrolora c, uvodi se ograničenje oblika $\texttt{penal}_c \leq \texttt{cena}$ (za fiksiranu vrednost neke celobrojne promenljive `cena`). Cilj je naći minimalnu nenegativnu vrednost ove promenljive uz zadovoljenje svih tvrdih ograničenja (naći najmanji maksimalni penal svih kontrolora), tj. učiniti najnezadovoljnijeg kontrolora što je moguće više zadovoljnim. Promenljivu `cena` zovemo *promenljiva funkcije cilja*, tj. u ovoj situaciji je funkcija cilja definisana sa

$f(x) = x$. Primetimo da sve promenljive $x_{c,i}$ imaju domen $\{0, 1\}$, pa se navedeno ograničenje može modelirati bilo kao linearni izraz (za CSP i linearni model) ili bulovsko ograničenje kardinalnosti (za bulovski model).

Koristimo 3 optimizacione tehnike koje su zasnovane na asimetričnoj binarnoj pretrazi. Pretraga je asimetrična jer se u zavisnosti od vrednosti promenljive k, favorizuju ili zadovoljive ili nezadovoljive instance. Za $k = 1/2$ ovo je simetrična binarna pretraga, a za $k > 1/2$ zadovoljive instance su favorizovane (obično su lakše za rešavanje). U našim preliminarnim eksperimentima se vrednost $k = 4/5$ pokazala dobrom, pa nju koristimo u svim eksperimentima koji se odnose na pravljenje rasporeda kontrolora letenja.

Opšti algoritam za sve 3 tehnike je prikazan na slici 4.1, a u nastavku su objašnjene razlike između tehnika. Promenljive `cena_l` i `cena_d` se inicijalizuju na početku algoritma i one predstavljaju trenutne granice promenljive `cena`. U svakoj iteraciji petlje se traži rešenje problema sa vrednošću maksimalnog penala koja je manja ili jednaka od trenutne vrednosti promenljive `cena` – generiše se nova instanca za odgovarajuću vrednost promenljive `cena` i potom rešava novim pokretanjem pridruženog rešavača. Proces rešavanja svaki put kreće od početka na instanci koja se razlikuje od prethodne instance samo u toj vrednosti. Ako je pronađeno rešenje (nađena vrednost promenljive `cena` je `obj`, gde je $\texttt{obj} = \max_{c=1}^{n_c} \texttt{penal}_c$ maksimalni penal kontrolora, a vrednost $\texttt{penal}_c$ je izračunata za nađene vrednosti $x_{c,i},\ i \in \{1, \ldots, m_c\}$), onda se pokušava sa poboljšanjem pronađenog rešenja. Upravo na tom mestu se razlikuju 3 tehnike, u zavisnosti od postupka koji koriste za poboljšavanje rešenja. Pronađeno (i potencijalno poboljšano) rešenje se pamti kao najbolje pronađeno do tog trenutka, kao i nova najbolja vrednost promenljive `cena`. U zavisnosti da li je pronađeno rešenje ili ne, ažurira se ili `cena_d` ili `cena_l`. Pretraga se završava i optimum je pronađen kada `cena_l` postane veća od `cena_d` (optimum postoji pod uslovom da postoji barem jedno rešenje).

Svesni smo da postoje pristupi koji mogu da unaprede efikasnost korišćenjem inkrementalnog rešavanja [46, 126], ali unapređenja ne bi trebalo da budu značajna zbog rezultata izvršenih eksperimenata (poglavlje 4.1.5.2). Linearna pretraga, binarna pretraga i mnogi drugi algoritmi [19] mogu biti korišćeni u traženju optimalne vrednosti promenljive `cena`.

`specifikacija_problema` – specifikacija problema koji treba rešiti
`maks_cena` – maksimalna vrednost promenljive `cena`

```
function ATCoSSP (specifikacija_problema, maks_cena)
  cena_l = 0;
  cena_d = maks_cena;
  while cena_l ≤ cena_d
    cena = cena_l + 4/5 · (cena_d − cena_l);
    instanca = Generiši (specifikacija_problema, cena);
    if (Pokreni_rešavač (instanca, &obj, &rešenje == SAT)
      Poboljšaj_rešenje (instanca, &obj, &rešenje);
      najbolji_obj = obj;
      najbolje_rešenje = rešenje;
      cena_d = najbolji_obj −1;
    else
      cena_l = najbolji_obj +1;
  return najbolje_rešenje;
```

Slika 4.1: Opšta tehnika rešavanja problema *ATCoSSP*.

4.1.4.1 *Osnovna* tehnika

Prva tehnika uopšte ne koristi poboljšanje rešenja, tj. odgovara kodu na slici 4.1 u kome je izostavljen poziv funkcije `Poboljšaj_rešenje`. Ova tehnika će služiti kao reper za poređenje drugih tehnika.

4.1.4.2 Tehnika *ZamenaSmena*

Kada je pronađeno rešenje sa nekim penalom `obj`, lokalna pretraga se koristi da bi se unapredilo rešenje (da se smanji vrednost `obj`). Lokalna pretraga se zasniva na zameni smena. Dve smene mogu da budu zamenjene između dva kontrolora ako su iste dužine i ako se zamenom ne krši nijedno od tvrdih ograničenja. Cilj ove tehnike je da se oponašaju razmene koje bi bile napravljene pri ručnom generisanju i poboljšavanju rasporeda. U primeru rasporeda datom u tabeli 4.1, ako Ana i Vesna imaju licence za iste pozicije i Ana preferira da radi ujutru a Vesna uveče, onda su njihove smene na dan 4 potencijalne smene za razmenu. Smene se razmenjuju u slučaju da veći od penala kontrolora nije uvećan posle zamene (nezadovoljniji kontrolor ne sme da postane još nezadovoljniji). Ovaj uslov osigurava da se vrednost `obj` ne povećava. Može se desiti da razmenom jedan ili više kontrolora postane nezadovoljniji a niko ne postane zadovoljniji. Ipak, ovo dozvoljavamo kako bi povećali broj isprobanih re-

šenja. Da bi se izbeglo zaglavljivanje u lokalnom minimumu, posle nekog fiksiranog broja iteracija, određeni broj slučajnih zamena smena se izvršava (time se možda povećava vrednost `obj`). Posle fiksiranog broja iteracija lokalne pretrage, binarna pretraga se nastavlja.

Tabela 4.1: Mali primer rasporeda smena za 4 dana (nije predstavljen raspored popunjenosti pozicija). Smene su 1 (04-12), 2 (08-16), 3 (12-20), 4 (relaksacija), 5 (odmor).

Ime	Dan 1	Dan 2	Dan 3	Dan 4
Ana	2 (08-16)	4 (relaks.)	2 (08-16)	3 (12-20)
Branko	4 (relaks.)	3 (12-20)	3 (12-20)	4 (relaks.)
Vesna	1 (04-12)	4 (relaks.)	1 (04-12)	2 (08-16)
Goran	4 (relaks.)	3 (12-20)	5 (odmor)	5 (odmor)

4.1.4.3 Tehnika *BezPozicija*

Cilj nam je da u procesu poboljšanja rešenja koristimo značajno manje modele – to se postiže zamenom ograničenja koja se odnose na pozicije ograničenjima koja se odnose na smene. Kao što je rečeno u poglavlju 4.1.3, određeni broj kontrolora je potreban za svaku poziciju u svakom satu. U pronađenom rešenju pre poboljšanja, dovoljan broj kontrolora je dodeljen svakoj poziciji u svakom satu i u isto vreme određen broj kontrolora je dodeljen svakoj smeni svakog dana (na primer, na dan 2 u primeru datom u tabeli 4.1, 2 kontrolora su dodeljena smeni 3 i nijedan kontrolor nije dodeljen smeni 1). Proces poboljšanja rešenja se sastoji iz dve faze i sledi opis te dve faze.

Smanjivanje broja dodeljenih kontrolora smenama. Cilj je smanjiti broj dodeljenih kontrolora određenim smenama i dobiti manje popunjen raspored on onog koji je nađen pre poboljšanja. To se postiže nalaženjem dva dana kada je isti broj kontrolora potreban za svaku poziciju u svakom satu, a tih dana radi različit broj kontrolora – u toj situaciji u jednom od ta dva dana se može smanjiti broj kontrolora koji tog dana dolazi na posao kopiranjem rasporeda smena iz drugog dana[8].

[8]U praksi je na aerodromima zaposlen veći broj kontrolora od minimalnog broja potrebnog za izvršenje svih zadataka u svim smenama (uzimajući pri tome u obzir da kontrolori odlaze i na odmor). Ovime je omogućeno da više želja zaposlenih bude ispunjeno kao, a izbegava se i preopterećenost kontrolora.

Prva pretpostavka je da je za neke dane d_1 i d_2 isti broj kontrolora potreban za svaku poziciju svakog sata ovih dana, što se često dešava u praksi (neka je ovo slučaj sa danima 1 i 3 u primeru). Druga pretpostavka je da je za svaku smenu broj dodeljenih kontrolora toj smeni na dan d_1 manji ili jednak od broja dodeljenih kontrolora istoj toj smeni na dan d_2. Treća pretpostavka je da za barem jednu smenu, broj dodeljenih kontrolora toj smeni na dan d_1 jeste striktno manji od onog na dan d_2 (dani 1 i 3 zadovoljavaju i drugu i treću pretpostavku). Ako su zadovoljene ove tri pretpostavke, onda je broj potrebnih kontrolora za svaku smenu na dan d_2 jednak odgovarajućem broju na dan d_1. U rasporedu se smene nekih kontrolora (postoji barem jedan takav kontrolor) koji rade na dan d_2 uklanjaju iz rasporeda dok se ne postigne da je svakoj smeni na dane d_1 i d_2 dodeljen isti broj kontrolora i pri tome se tim kontrolorima dodeljuje dan relaksacije na dan d_2, a dodela pozicija za dan d_1 se kopira u dodelu pozicija za d_2. Na taj način se smanjuje broj kontrolora dodeljenih smenama na dan d_2 (nijedan kontrolor nije potreban za smenu 3 na dan 3). Prethodni postupak ponavljamo sve dok ne postoji dan za koji je moguće smanjiti broj potrebnih kontrolora za neku smenu. Napomenimo da se navedenim postupkom neka ograničenja mogu učiniti privremeno nezadovoljenim, tj. raspored koji se dobija nije ispravan (na primer, neki kontrolor sada može imati veći broj uzastopnih dana relaksacije od dozvoljenog). Cilj ovog dela je da se izračuna koliko je kontrolora potrebno za koju smenu.

Binarna pretraga na redukovanom modelu. Poboljšanje rešenja se sada nastavlja unutrašnjom binarnom pretragom sa modelima u kojima su pozicione promenljive i ograničenja zamenjeni ograničenjima (generišu se nove instance) koja određuju minimalan broj neophodnih kontrolora za svaku smenu svakog dana (na primer, u CSP modelu za dan 2: $count(\{dc_{2,1}, \ldots, dc_{2,4}\},\ 3) \geq 2$ – barem 2 kontrolora rade smenu 3). Svaki sat ovih smena je već povezan sa pozicijom (ovo je opisano u prethodnom paragrafu). Ovo značajno smanjuje veličinu modela (originalni model ćemo zvati *kompletni*, a ovaj model *redukovani* model). Kada se optimum na redukovanom modelu pronađe, to nije obavezno i optimum početnog problema (tj. kompletnog modela). Poboljšanje rešenje je završeno i binarna pretraga se potom nastavlja na kompletnom modelu.

4.1.5 Eksperimentalna evaluacija

Svi testovi su vršeni na istom računaru koji je već opisan u poglavlju 2.5.2. Korišćeno je vremensko ograničenje od 600 minuta (10 sati) po instanci, uključujući i generisanje ulaznih datoteka i rešavanje (prvo je zanemarljivo u poređenju sa drugim).

Instance. Eksperimentisali smo sa pravljenjem mesečnog rasporeda za 2 različita meseca na aerodromu u Vršcu, koji je u vreme pravljenja rasporeda zapošljavao 12 kontrolora. Postojala je potreba za isprobavanjem raznih ulaznih parametara, da bi se izabrao raspored koji je najpogodniji. Na primer, za isti mesec su pravljeni rasporedi sa različitim brojem (od 3 do 5) smena i njihovim radnim satima, kao i sa različitim brojem kontrolora kojima je dozvoljeno da uzmu odmor. Za svaki mesec, generisali smo 9 ulaznih instanci. Jedno od 9 rešenja je izabrano da bude formalni raspored[9]. Na ovaj način smo generisali 18 instanci iz realnog okruženja. Dodatno, generisali smo 4 teže instance da bi procenili skalabilnost različitih metoda rešavanja. Ove instance obuhvataju više kontrolora (17-25), duže periode (60-120 dana) i veći broj neophodnih kontrolora za pojedine pozicije u određenim satima.

Tabela 4.2 prikazuje neke karakteristike instanci kod sva 3 modela i za kompletan i za redukovani model korišćen tokom lokalne pretrage u tehnici *BezPozicija*. Brojevi pokazuju da je veličina redukovanog modela značajno manja u odnosu na kompletan model. Umanjenje je posledica velikog broja promenljivih i ograničenja koje se odnose na zahteve vezane za pozicije. Te promenljive i ograničenja moraju biti uvedene za svakog kontrolora, za svaki dan i sat, pa smatramo da veličina kompletnog modela ne može biti bitno smanjena. Iako se prva dva modela razlikuju samo u načinu na koji su neka ograničenja modelirana, CSP model je znatno kompaktniji jer koristi globalna ograničenja.

4.1.5.1 Metode rešavanja i preliminarni eksperimenti

Metode rešavanja. Sa svakom od tehnika može se koristiti veliki broj metoda rešavanja. Četiri potpune metode rešavanja su iskorišćene za pravljenje rasporeda.

Prva metoda koristi savremene ne-SAT-orijentisane CSP rešavače za rešavanje generisanih CSP i COP instanci u dva formata. Specifikacije u prva dva modela

[9]Najvažniji parametar u odlučivanju je bio broj kontrolora kojima je dozvoljeno da uzmu odmor – cilj je bio maksimizovati ovaj broj.

Tabela 4.2: Prosečan broj promenljivih i ograničenja, kao i prosečna veličina domena na svim instancama problema *ATCoSSP*. Instance su generisane tokom korišćenja tehnike *BezPozicija*.

	Promenljive		Ograničenja		Domen	
Model	kompletni	redukovani	kompletni	redukovani	kompletni	redukovani
CSP	10215	3951.7	81295.3	13400	4.79	2.79
Linearni	35222.2	4467	131840	14430	2.82	2.70
Bulovski	55447.9	12964	160314	65293.1	2	2

mogu se direktno prevesti u ove formate. XCSP format je korišćen od strane rešavača *Mistral* 1.545 i *Abscon* 112v4. MiniZinc format je korišćen od strane rešavača `mzn-g12cpx`, `mzn-g12fd`, `mzn-g12lazy`, `mzn-g12mip` uključenih u MiniZinc 1.6 distribuciju i rešavača mzn-gecode Gecode-4.2.0. Svi navedeni rešavači su već opisani u poglavljima 2.2.4 i 2.3.8.

Druga metoda svodi CSP i COP instance u spomenutim formatima na probleme SAT, Partial MaxSAT i SMT (u teoriji linearne celobrojne aritmetike). U ovom pristupu, ulazne instance se prevode u instance pomenutih problema. Savremeni rešavači se koriste za nalaženje rešenja koja se potom prevode u rešenja originalnih problema CSP i COP. Sistemi Sugar v2-0-3 i Sat4j 2.3.4 [16] se koriste za svođenje na SAT. Kod svođenja na SMT su korišćeni rešavači Z3 v4.2 i Yices 2.2. Rešavači problema Partial MaxSAT QMaxSAT 0.4 [77] i MSUnCore-20130422 [92] su korišćeni za rešavanje (malo modifikovanih) instanci generisanih od strane sistema Sugar.

Treća metoda rešava instance bulovskog modela direktno navedene u ulaznim formatima problema PB i SAT. Koristili smo SAT rešavače *clasp* 2.1.3 [54], MiniSat 2.2 [45] i Lingeling aqw-27d9fd4-130429 [18], kao i PB rešavače *clasp* 2.1.3 i MiniSat+ 1.0 [47]. U svim predstavljenim rezultatima su za svođenje na klauze problema SAT i Partial MaxSAT korišćeni sekvencijalni brojači (jer su u našim eksperimentima bili efikasniji od mreža kardinalnosti sa kojima smo takođe eksperimentisali) implementiranih u sistemu *meSAT*. Za broj promenljivih većih od 20 u bulovskom ograničenju kardinalnosti najviše-jedan-tačan korišćeno je kodiranje proizvoda, dok je za broj promenljivih manji od 20 korišćeno komandant kodiranje (ova kombinacija je dala najbolje rezultate).

Četvrta metoda koristi ILP rešavač IBM ILOG CPLEX Optimization Studio[10] pri čemu se specifikacija u linearnom modelu direktno prevodi u ulazni format ovog rešavača.

[10] `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`

Preliminarni eksperimenti. Ovi eksperimenti su sprovedeni na 5 slučajno odabranih instanci sa ciljem da se eliminišu manje efikasni rešavači. Svi rešavači izuzev Partial MaxSAT rešavačima su korišćeni sa opisanim optimizacionim tehnikama, jer su se ove tehnike pokazale efikasnijim od ugrađenih optimizacionih algoritama rešavača. Svi rešavači su korišćeni u svojim podrazumevanim konfiguracijama.

4.1.5.2 Eksperimentalni rezultati

U ovom poglavlju predstavljamo rezultate samo za rešavače koji su postigli najbolje rezultate u preliminarnim eksperimentima i za *interesantne instance*, tj. za one instance na kojima nisu svi najbolji rešavači našli optimum u datom vremenskom ograničenju.

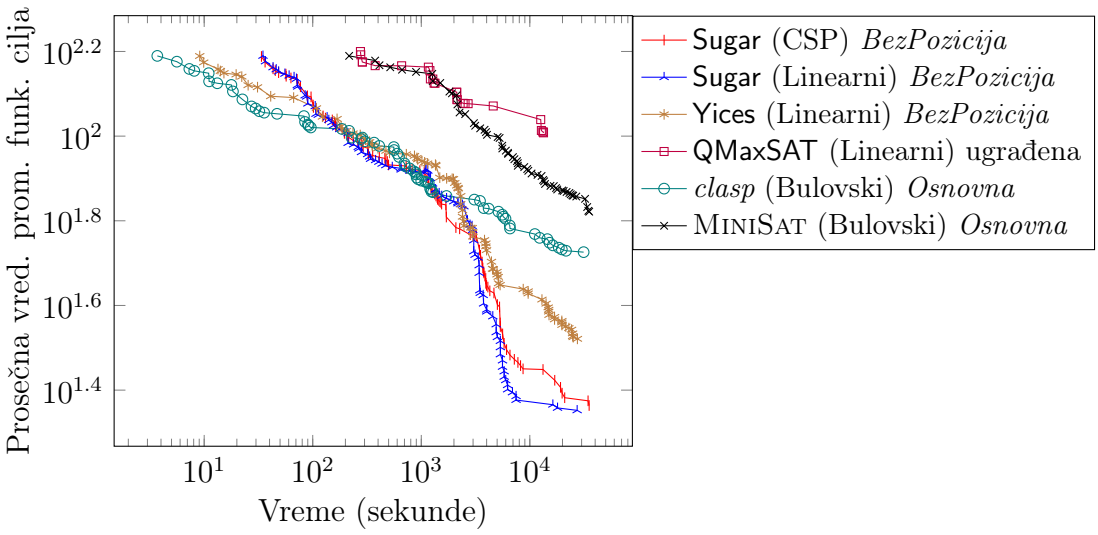
Poređenje tehnika. U pristupu *ZamenaSmena* smo koristili 10^6 iteracija. Posle svakih 10^4 iteracija, vršene su slučajne zamene smene. Ipak, nije bilo unapređenja u vrednosti `obj`, pa ne uključujemo ovaj pristup u predstavljene rezultate. Ova tehnika odgovara načinu poboljšanja rešenja kod ručno generisanog rasporeda i pokazuje da se jednom generisan raspored ručno vrlo teško unapređuje. Tabela 4.3 prikazuje rezultate poređenja između dve preostale tehnike korišćenjem najboljih rešavača. Kodiranje uređenja koje koristi Sugar je ipak bilo nešto efikasnije od onog implementiranog u sistemu *meSAT*, pa je zato *meSAT* izostavljen iz tabele rezultata. Rezultati pokazuju da u slučaju najboljih rešavača (Sugar i Yices) tehnika *BezPozicija* ima bolju efikasnost od tehnike *Osnovna*.

Tabela 4.3: Poređenje tehnika na interesantnim instancama problema *ATCoSSP*. (vremensko ograničenje je 600 minuta). Polja sadrže prosečne postignute vrednosti: što je manja vrednost, bolji rezultat je postignut; redovi predstavljaju tehnike; *clasp* je korišćen za rešavanje PB instanci.

Model	CSP	Linearni	Linearni	Bulovski	Bulovski
Metoda	Sugar	Sugar	Yices	*clasp*	MiniSat
Osnovna	31.5	29.3	50.8	53.2	66.3
BezPozicija	23.1	22.5	33.2	72.7	73.7

Detaljni rezultati. Tabela 4.4 pokazuje rezultate za najbolje kombinacije rešavača i tehnika. Instance 19-22 su dodatno generisane teške instance. Rezultati pokazuju da među najboljim rešavačima nije bilo ne-SAT-orijentisanih rešavača (*Abscon*

je bio jedini ne-SAT-orijentisan rešavač koji je uspeo da reši neke od instanci, ali nije uspeo da poboljša nijedno od nađenih rešenja). Interesatno, CPLEX nije uspeo da reši nijednu od instanci. Uklanjanjem ograničenja *Pozicije popunjene*, CPLEX je uspeo da reši neke od instanci pa su ova ograničenja najverovatniji razlog neefikasnosti. Pripisujemo uspeh SAT-orijentisanih rešavača malim domenima promenljivih i velikom broju ograničenja koja predstavljaju odnose između promenljivih (ona čine 75% generisanih instanci). Najbolji rezultati u proseku su postignuti svođenjem na SAT korišćenjem sistema Sugar. Ipak, postoje slučajevi kada su drugi rešavači ostvarili bolje performanse. Ako pogledamo teške instance (19-22), vidimo da Sugar ostvaruje značajno bolje performanse od ostalih rešavača. Pošto bulovska ograničenja kardinalnosti uključuju veliki broj promenljivih u bulovskom modelu, MiniSat i *clasp* su manje uspešni sa ovim instancama. Sugar efikasno obrađuje aritmetička ograničenja korišćenjem kodiranja uređenja. Ovaj rešavač može da obradi veliki broj ograničenja jer efikasno koristi ugrađeno propagiranje.



Slika 4.2: Prosečna vrednost promenljive funkcije cilja postignuta kroz vreme za problem *ATCoSSP* – svaki znak na krivama predstavlja jedno smanjivanje te vrednosti; korišćeni model za svaki rešavač je dato u zagradama.

Slika 4.2 prikazuje promenu prosečne vrednosti promenljive funkcije cilja postignute na interesantnim instancama tokom vremena (odgovara paralelnom pokretanju testova) za iste kombinacije rešavača i tehnika kao i u tabeli 4.4. Uzeto je da je vrednost promenljive funkcije cilja postignuta na početku 160 za sve metode rešavanja na svim instancama. Rešavač Sugar ostvaruje najbolje performanse i slične je efikasnosti i na CSP i na linearnom modelu. Ovo može biti pripisano sličnošću ovih

Tabela 4.4: Rezultati poređenja na interesantnim instancama problema *ATCoSSP* (vremensko ograničenje je 600 minuta). Redovi predstavljaju različite instance. Ispod imena svakog rešavača data je oznaka tehnike sa kojom je korišćen. Svako polje sadrži vrednost promenljive funkcije cilja, a vreme potrebno da bi se postigla ova vrednost je dato u zagradama. Kurzivnim slovima su u 2. koloni predstavljene postignute vrednosti promenljive funkcije cilja koje su jednake optimumima. Polja koja predstavljaju najbolju metodu za svaku instancu su popunjena podebljanim slovima. Broj 160 označava da rešavač nije našao nijedno rešenje na datoj instanci (160 je korišćen jer je veći od svih dobijenih vrednosti promenljive funkcije cilja). Za rešavanje PB instanci korišćen je *clasp*.

	Model	CSP	Linearni			Bulovski	
Instanca	Najbolja vred. promenljive funkcije cilja	Sugar *BezPoz.*	Sugar *BezPoz.*	Yices *BezPoz.*	QMaxSAT ugrađena optimiz.	*clasp* *Osnovna*	MiniSat *Osnovna*
2	52	61 (1)	55 (1)	**52 (1)**	160 (0)	54 (51)	52 (35)
3	52	55 (3)	60 (1)	54 (8)	160 (0)	58 (0)	**52 (129)**
4	28	34 (8)	**28 (41)**	36 (413)	160 (0)	36 (99)	38 (538)
5	*6*	**6 (54)**	10 (455)	12 (457)	160 (0)	20 (95)	10 (587)
6	*20*	**20 (28)**	28 (272)	20 (77)	20 (45)	20 (70)	20 (342)
10	*16*	16 (137)	20 (6)	22 (15)	26 (11)	22 (2)	**16 (105)**
14	*0*	6 (78)	0 (36)	0 (37)	0 (223)	**0 (21)**	0 (447)
15	*4*	20 (587)	18 (125)	**4 (418)**	160 (0)	42 (109)	66 (577)
18	*12*	13 (88)	20 (18)	14 (414)	**12 (77)**	15 (242)	18 (411)
19	*8*	**8 (25)**	8 (40)	8 (329)	160 (0)	10 (522)	110 (563)
20	8	14 (220)	15 (67)	**8 (329)**	160 (0)	95 (109)	160 (0)
21	12	13 (573)	**12 (300)**	42 (145)	160 (0)	160 (0)	160 (0)
22	18	34 (89)	**18 (125)**	160 (0)	160 (0)	160 (0)	160 (0)
Prosek	18.2	23.1	22.5	33.2	115.2	53.2	66.3

modela, sa razlikom modeliranja samo nekih ograničenja. Svođenje na PB postiže dobre rezultate na početku rešavanja i predstavlja najbolju metodu rešavanja za nalaženje brzih rešenja. Ipak, performanse ove metode nisu dovoljno dobre sa povećanjem utrošenog vremena. Prosečan broj pokretanja rešavača je 19 u slučaju kada je pronađen optimum, inače je 3. U predstavljanju sistema Sugar++ [126] (verzija sistema Sugar koja koristi inkrementalno rešavanje) na takmičenju "Third International CSP Solver Competition"[11], autori su naglasili da se vreme rešavanja inkrementalne pretrage može značajno unaprediti kada je broj pokretanja rešavača mali (manji od 6). Zbog ovih razloga mislimo da inkrementalna pretraga ne može

[11] `http://www.cril.univ-artois.fr/CPAI08/`

značajno da unapredi pronađene vrednosti promenljive funkcije cilja, iako bi mogla da smanji utrošeno vreme u slučajevima kada je optimum pronađen.

Korišćenje *ATCoSSP* instanci na takmičenjima rešavača. Na takmičenjima rešavača se svake godine koristi podskup instanci od prethodnih godina dopunjen novim instancama aktuelnih problema. Ovime se postiže da su korpusi raznovrnsni i reprezentativni. Instance *ATCoSSP* problema koje smo generisali su korišćene na dva takmičenja: takmičenju SAT rešavača SAT Competition 2014 i takmičenju MaxSAT rešavača[12].

4.1.6 Pravljenje rasporeda u praksi

Rasporedi za aerodrom u Vršcu su generisani ručno pre korišćenja tehnika opisanih u ovoj glavi. Potreba za automatskim generisanjem rasporeda je sezonska. Broj potrebnih radnih sati tokom letnjih meseci se povećava a broj dostupnih kontrolora se smanjuje, pošto je to period kada najviše zaposlenih ide na odmor. Zato postaje veoma teško ručno generisati rasporede za ove mesece. Tokom letnjih meseci 2013, pravljenje rasporeda je korišćeno kao usluga, gde je autor teze generisao rasporede na osnovu zahteva zaposlenih.

Generisali smo dva rasporeda ručno, da bi ih uporedili sa onima koji su automatski generisani. U proseku, postigli smo vrednost promenljive funkcije cilja 50 tokom 3 sata ručnog generisanja i vrednost 28 tokom 11 minuta automatskog generisanja (Sugar (2) *BezPozicija*). Mnoge želje kontrolora izražene pomoću mekih ograničenja postale su ostvarive automatizacijom. Veći broj ispunjenih želja vodi i većem zadovoljstvu zaposlenih. Pošto automatski generisani rasporedi smanjuju opterećenje po kontroloru, zaposleni se manje zamaraju i time se dodatno povećava bezbednost. Takođe, ručno pravljenje rasporeda je bilo vrlo podložno greškama. Nije moguće napraviti značajno unapređenje jednom generisanog rasporeda (u pogledu ostvarenih želja) korišćenjem jednostavnih zamena smena, što je i rečeno u komentaru rezultata tehnike *ZamenaSmena* (deo 4.1.5.2). Rešenje se ručno potencijalno može unaprediti ako se istovremeno razmatra veći broj kontrolora i smene različitih dužina, ali je ovo u našem iskustvu bilo skoro nemoguće postići u praksi.

[12]Linkovi na takmičenja su dostupni na veb-strani: `http://baldur.iti.kit.edu/sat2014/competitions.html`

4.2 Generisanje i rešavanje velikih sudoku zagonetki

Poboljšanje pristupa za rešavanje zagonetki, igara i kombinatornih zadataka obično povlači i uspešniju primenu ovih pristupa u rešavanju realnih problema. Zato postoji veliko interesovanje za generisanje teških zagonetki, igara i kombinatornih zadataka i sve tri vrste problema su često deo poznatih i dosta korišćenih sistema, kao što su Minizinc 1.6 [98], Sugar [124], itd. Već je spomenuto da u mnogim oblastima organizuju takmičenja rešavača (na primer, SAT i CSP takmičenja) da bi se odredilo koji rešavač je efikasniji i samim tim verovatno pogodniji za rešavanje realnih problema. Zagonetke, igre i kombinatorni zadaci obično čine značajan deo korpusa instanci na kojima se vrši poređenje rešavača koji su učesnici tih takmičenja. Zato u procesu evaluacije rešavača na takmičenjima možemo razlikovati dve faze: generisanje instanci na kojima će se vršiti poređenje rešavača i samo poređenja tih rešavača.

Sudoku zagonetke su doživele veliku popularnost u poslednjih nekoliko decenija. Zagonetka se sastoji od table dimenzije $n \times n$ koja je podeljena u blokove dimenzija $m \times m$ ($m \times m = n$) na kojoj se nalaze početni brojevi koji se ne mogu ni menjati ni pomerati. Zagonetka je rešena kada je tabla popunjena tako da svaki red, kolona i blok sadrže različite brojeve od 1 do n.

Razvijen je veliki broj metoda za rešavanje sudoku zagonetki pomoću papira i olovke. Takođe, postoji na desetine (ako ne i stotine) radova na temu rešavanja ovih zagonetki pomoću računara. Skoro svi radovi se odnose na rešavanje zagonetki dimenzije 9×9. Iako ove zagonetke mogu biti vrlo zahtevne kada se rešavaju pomoću papira i olovke, čak i jednostavni bektreking algoritmi na savremenim računarima rešavaju i najteže među njima za nekoliko sekundi. Postoji samo nekoliko pristupa razvijenih za rešavanje sudoku zagonetki dimenzija većih od 9×9 i mi ćemo se u ovoj glavi baviti samo ovim zagonetkama. Prikaz i rezultati koji slede su velikim delom zasnovani na objavljenom radu autora teze "Generisanje i rešavanje velikih Sudoku zagonetki svođenjem na SAT problem" [121].

Među postojećim metodama za rešavanje sudoku zagonetki dimenzija većih od 9×9 razlikujemo one zasnovane na egzaktnim metodama [34, 76, 80] i metaheuristikima [81, 83]. Jedan od načina rešavanja sudoku zagonetki je rešavanje svođenjem na SAT. Najpoznatije svođenje na SAT sudoku zagonetki dimenzije 9×9 su razvili Linc sa koautorima [82]. Ovo svođenje je dodatno unapređeno korišćenjem tehnika

propagiranja ograničenja i sudoku zagonetke različitih dimenzija su korišćene za testiranje [76]. Na žalost, autori nisu opisali svojstva korišćenih zagonetki, a efikasnost upotrebljenog načina rešavanja nije upoređena ni sa jednom drugom metodom (rešavanje sudoku zagonetki nije u fokusu tog rada). Simulirano kaljenje je korišćeno za rešavanje zagonetki dimenzija do 25×25 [81], a ova metoda je u novom radu i unapređena [83]. Za rešavanje sudoku zagonetki dimenzija do 16×16 korišćeno je i paralelno izvršavanje na više procesora istovremeno [34]. Verovatnosni grafovski modeli su upotrebljavani za rešavanje sudoku zagonetki dimenzija do 16×16 [74].

Pre poređenja metoda na sudoku zagonetkama, potrebno je generisati skup pogodnih zagonetki. Kao što je već pomenuto, zbog vremena rešavanja koje se obično meri delovima sekunde, poređenje metoda na zagonetkama dimenzije 9×9 ne daje pravu sliku efikasnosti tih metoda. Generisanje teških zagonetki većih dimenzija je otvoren problem i vrlo mali broj radova se do sada bavio ovom problematikom. Poznate su nam dve metode za generisanje velikih sudoku zagonetki. Kod prve [81] se generišu zagonetke različitih nivoa popunjenosti ali ne postoji garancija jedinstvenosti rešenja. Druga metoda [34] garantuje jedinstvenost rešenja generisanih zagonetki.

Generisanje teških sudoku zagonetki je od interesa i za sisteme opšte namene, koji rešavaju mnogo širi skup problema od sudoku zagonetki. Na primer, da bi se poredila efikasnost algoritama razvijenih za globalno ograničenje *all-different*, potrebne su instance problema koji se modeliraju pomoću velikog broja pojavljivanja ovog ograničenja. Sudoku se lako i prirodno modelira samo pomoću *all-different* ograničenja o čemu je još 2005. godine pisao Simonis u jednog od prvih radova na temu modeliranja i rešavanja sudoku zagonetki pomoću programiranja ograničenja [114]. Naime, uvode se n^2 promenljivih sa istim domenom $\{1, \ldots, n\}$ gde svaka promenljiva odgovara jednom polju zagonetke i postavljaju se ograničenja da u svakom redu/koloni/bloku pripadajuće promenljive moraju uzimati međusobno različite vrednosti. Stoga je sudoku pogodan problem za poređenje različitih algoritama koji se odnose na ovo ograničenje (na primer, pogledati rad Bankovića [10]).

4.2.1 Rešavanje sudoku zagonetki svođenjem na SAT i preprocesiranje

Najpoznatije svođenje na SAT sudoku problema [76] se sastoji u tome da se za svako polje (r,k) (r i k su red i kolona, tj. $1 \leq r \leq n$, $1 \leq k \leq n$) i za svaku moguću vrednost v ($1 \leq v \leq n$) koja se smešta u zagonetku, uvodi iskazna promenljiva $x_{r,k,v}$ (tačna je, tj. uzima vrednost 1 akko je vrednost v smeštena na polje (r,k)). Skup ovako uvedenih promenljivih ćemo označiti sa X. Promenljive koje se odnose na početno popunjena polja i promenljive koje ne mogu da budu tačne jer je vrednost kojoj odgovaraju već smeštena u isti red/kolonu/blok se eliminišu (obe ove grupacije promenljivih imaju svoju istinitosnu vrednost već određenu). Koristeći ovu eliminaciju, ostaje skup preostalih (nepoznatih) promenljivih koji ćemo označiti sa Y (svaka preostala promenljiva $x_{r,k,v}$ je preimenovana u $y_{r,k,v}$). Cilj eliminacije promenljivih je da se smanji veličina instance koja će se proslediti SAT rešavaču, što može značajno ubrzati vreme rešavanja. Autori navedenog svođenja u svom prethodnom radu [80] navode da se ovom eliminacijom broj klauza može smanjiti i do 79 puta.

U cilju dodatne eliminacije promenljivih, uvodimo vrlo jednostavna pravila preprocesiranja koja će biti primenjena na zagonetku, pre uvođenja iskaznih promenljivih. Pravila odgovaraju zaključivanju koje se vrši pri rešavanju zagonetki pomoću papira i olovke. Za svako nepopunjeno polje, održava se lista svih vrednosti koje je moguće staviti na to polje. Primenjujemo sledeća pravila:

1. Jedna preostala vrednost. Vrednost svakog polja na koje može da se smesti samo jedna vrednost v je određena, i vrednost ovog polja se postavlja na v.

2. Jedno preostalo polje. Ako se neka vrednost v može smestiti na tačno jedno polje u nekom redu/koloni/bloku, onda se vrednost ovog polja postavlja na v.

3. Putna konzistencija (termin potiče iz rada [83] koji je već koristio ovo pravilo). Ako tri polja u istoj vrsti/koloni/bloku imaju svojstvo da se na dva od njih mogu smestiti samo vrednosti v_1 i v_2, a na treće samo vrednosti v_1, v_2 i v_3, onda se u treće polje smešta vrednost v_3 jer se vrednosti v_1 i v_2 moraju smestiti u prva dva polja.

Iterativno primenjujući navedena pravila dok se ni jedno pravilo više ne može primeniti u zagonetki, a tek onda primenjujući opisanu eliminaciju promenljivih,

dobija se još manji skup nepoznatih promenljivih Z, u odnosu na prethodno opisani skup Y (svaka preostala promenljiva $y_{r,k,v}$ je preimenovana u $z_{r,k,v}$). Ceo ovaj postupak se smatra preprocesiranjem.

Na preostalim promenljivima Z, opisanim u prethodnom pasusu, uvode se bulovska ograničenja kardinalnosti. Preciznije, ograničenja koja se uvode i koja predstavljaju svođenje na SAT su:

1. Za svako nepopunjeno polje, tačno jedna od pridruženih promenljivih kojima još nije određena vrednost je tačna.

2. Za svaku vrednost i svaki red/kolonu/blok gde ta vrednost još nije smeštena, tačno jedna od pridruženih promenljivih toj vrednosti u tom redu/koloni/bloku je tačna. Ovo ograničenje se postavlja samo nad promenljivama koje još nemaju dodeljenu vrednostu u redu/koloni/bloku, tj. samo na promenljivama skupa Z. Na primer, za sudoku dimenzija 16×16, vrednost 3 i red 1, pri čemu u prvom redu samo polje 1 ima dodeljenu vrednost 2, postavlja se ograničenje $z_{1,2,3} + \ldots + z_{1,16,3} = 1$ (3 se nalazi na tačno jednom mestu u prvom redu), tj. promenljiva koja se odnosi na polje (1,1) se preskače pošto već ima dodeljenu vrednost 2.

Kao što je već rečeno, dobijena SAT formula se prosleđuje SAT rešavaču koji izračunava vrednosti promenljivih u slučaju zadovoljivosti formule (zagonetka ima rešenja) ili vraća odgovor da formula nije zadovoljiva (zagonetka nema rešenja).

4.2.2 Generisanje sudoku zagonetki

Postojeći algoritam za generisanje zagonetki [34] polazi od rešenja zagonetke koje zovemo početno rešenje. Ovo rešenje se više puta može koristiti kao ulaz koristeći permutacije redova i kolona, permutacije kvadratnih redova i kolona (sastoje se iz blokova) i rotacijom table (ova pravila već su opisivana u ranijim radovima [34, 83]). Algoritam na slučajan način bira polje (r, k) i uklanja vrednost iz tog polja. Izmenjena zagonetka se rešava, sa dodatnim ograničenjem da se uklonjena vrednost ne može staviti na polje (r, k). Ako se pronađe rešenje izmenjene zagonetke, onda se uklanjanjem vrednosti broj rešenja povećao na barem dva (početno i pronađeno), pa se vrednost vraća u polje da bi se zadržala jedinstvenost rešenja. Inače, vrednost ostaje uklonjena, bez uticaja na jedinstvenost rešenja. Ovaj proces se ponavlja dok se zagonetka ne isprazni do nivoa popunjenosti kada se uklanjanjem bilo koje

vrednosti gubi jedinstvenost rešenja. Na ovaj način generišu se vrlo teške zagonetke [34]. Primetimo da jedno generisanje podrazumeva veliki broj rešavanja.

U cilju ubrzanja generisanja zagonetki, dodajemo neke korake pre opisanog postupka. Ovi koraci predstavljaju prvi deo algoritma generisanja, a neznatno modifikovani postupak opisan u prethodnom pasusu drugi deo. Algoritam koji smo razvili zovemo Modifikovano generisanje.

Pseudokod prvog dela generisanja je prikazan na slici 1. U prvoj petlji se na slučajan način uklanjaju vrednosti iz kopije početnog rešenja, sve dok se procenat popunjenih polja ne smanji na neku vrednost `t` (na primer, 20%). U ovom trenutku, skoro uvek postoji više rešenja zagonetke[13]. U drugoj petlji zagonetka se rešava uz uslov da se traži rešenje različito od početnog. Ako je novo rešenje pronađeno, lociraju se polja koja sadrže vrednosti različite od vrednosti početnog rešenja. Da bi se zabranilo novo rešenje, ova polja se popunjavaju vrednostima početnog rešenja. Koraci rešavanja zagonetke uz zabranu početnog rešenja i popunjavanja novim vrednostima se ponavljaju dok početno rešenje ne postane jedinstveno rešenje zagonetke (u generisanjima koje smo mi izvršavali procenat popunjenosti zagonetke u ovom trenutku generisanja je bio oko 50%).

```
Generiši_sudoku_sa_jedinstvenim_rešenjem (rešenje)
  Kopiraj (zagonetka, rešenje);
  while (Izračunaj_procenat_popunjenosti (zagonetka) > t)
    (i, j) = Vrati_slučajno_odabranu_popunjenu_poziciju (zagonetka);
    zagonetka [i][j] = prazna_pozicija;
  while (1)
    novo_rešenje = Reši_zabranjujući_rešenje (zagonetka, rešenje);
    if (novo_rešenje == UNSAT)
      break;
    Popuni_polja_koja_se_razlikuju (zagonetka, rešenje, novo_rešenje);
  return (zagonetka, rešenje);
```

Slika 4.3: Generisanje zagonetke sa jedinstvenim rešenjem – na ulazu je popunjena tabla koja će predstavljati jedinstveno rešenje zagonetke koja će biti generisana.

Prikazani algoritam je opšti i može se koristiti sa bilo kojom metodom rešavanja. U slučaju svođenja na SAT, funkcija `Reši_zabranjujući_rešenje` sastoji se od već opisanog uvođenja iskaznih promenljivih i preprocesiranja, generisanja SAT klauza,

[13]Prilikom vršenja eksperimenata čiji su rezultati prikazani u narednom poglavlju, vrlo su retki bili slučajevi kada nije postojalo više rešenja, i u tim slučajevima potrebno je dodatno isprazniti zagonetku.

dodavanja klauze koja zabranjuje početno rešenje i pokretanja SAT rešavača. Na primer, ako su prazna polja na tabli samo (1,1), (2,2) i (3,3) a početno rešenje na ove pozicije smešta redom 4, 7 i 9, onda bi klauza koja zabranjuje to rešenje bila $\neg z_{1,1,4} \vee \neg z_{2,2,7} \vee \neg z_{3,3,9}$ (barem jedna od vrednosti 4, 7 i 9 nije na odgovarajućem polju, tj. novo rešenje na barem jednoj poziciji mora da se razlikuje od početnog, što je u ovom slučaju nemoguće).

Posle prvog dela algoritma primenjuje se drugi deo koji je već opisan na početku ovog poglavlja. Jedina modifikacija je da se polje (r, k) bira prolazeći redom kroz sva popunjena polja jedno po jedno, a ne na slučajan način kao u originalnom algoritmu. Na ovaj način se garantuje da će generisana zagonetka biti što je moguće manje popunjena. Cilj modifikacije je da se rešavanjem manjeg broja zagonetki ubrza proces generisanja. Pseudokod drugog dela generisanja je prikazan na slici 2.

```
Isprazni_sudoku_do_manje_popunjenog (zagonetka, rešenje)
  while ( ((i, j) = Vrati_sledeće_popunjeno_polje (zagonetka)) != kraj)
    zapamćena_vrednost = zagonetka [i][j];
    zagonetka [i][j] = prazna_pozicija;
    novo_rešenje = Reši_zabranjujući_rešenje (zagonetka, rešenje);
    if (novo_rešenje != UNSAT)
      zagonetka [i][j] = zapamćena_vrednost;
  return (zagonetka, rešenje);
```

Slika 4.4: Kreiranje manje popunjene zagonetke sa jedinstvenim rešenjem.

4.2.3 Eksperimentalni rezultati

Svi testovi su vršeni na višeprocesorskom računaru sa 4 procesora AMD Opteron(tm) CPU 6168 na 1.9Ghz (svaki sa po 1 jezgara), sa 2GB RAM memorije po jezgru, pod operativnim sistemom Ubuntu server 2.6.32-38. Ograničenja kardinalnosti su svedena na SAT korišćenjem komandant kodiranja [76] implementiranih u okviru sistema meSAT [122]. Rešavač MiniSat 2.2 [45] je korišćen za rešavanje SAT instanci.

Nadalje će se termin rešavanje svođenjem na SAT odnositi na prevođenje problema sudoku u SAT formulu i pokretanje SAT rešavača MiniSat.

4.2.3.1 Eksperimenti koji se odnose na rešavanje

Najveći problem pri poređenju sa postojećim pristupima koji su razvijeni za rešavanje velikih sudoku zagonetki je da najčešće ni izvorni kod a ni korišćene zago-

netke nisu javno dostupni. Da bismo omogućili drugima poređenje sa našim pristupom, sistem je zajedno sa svim korišćenim zagonetkama javno dostupan na adresi `http://jason.matf.bg.ac.rs/~mirkos/Sudoku.html`.

Generisali smo instance na isti način kao u već postojećem radu Maćada sa koautorima [83]. U tom radu je heuristički pristup koji je koristio metaheuristiku *simulirano kaljenje* ostvario najbolje rezultate. Počevši od rešene zagonetke i fiksirane vrednosti p, generisanjem se vrednost svakog polja uklanja sa verovatnoćom p i na taj način dobija zagonetka koju treba rešiti. Generisanja se pokreću za različite vrednosti p, pri čemu p uzima vrednosti između 0.05 i 1.0, sa korakom 0.05. Što je vrednost p veća, to je manji broj popunjenih polja u zagonetki (1.0 odgovara potpuno praznoj zagonetki). Počevši od različitih rešenih zagonetki, za svaku od 20 vrednosti p generisali smo po 20 zagonetki. Na ovaj način je generisano po 400 zagonetki za svaku od dimenzija 16×16, 25×25 i 36×36. Tabela 4.5 prikazuje rezultate poređenja originalnog, najpoznatijeg rešavanja svođenjem na SAT [76] sa našom verzijom u kojoj je korišćeno preprocesiranje pre svođenja. Rezultati pokazuju da ne postoji bitna razlika između originalnog svođenja i onog koje koristi preprocesiranje. Rešavanje originalnim svođenjem je čak rešilo jednu instancu više dimenzije 36×36, ali pošto je razlika vrlo mala, ona se može pripisati slučajnim varijacijama u rešavanju pomoću SAT rešavača.

Izvorni kod pristupa koji koristi simulirano kaljenje nije javno dostupan i naš cilj je bio da prevaziđemo problem poređenja korišćenjem slabijeg računara i velikog broja zagonetki generisanih na isti način kao u originalnom radu (ovo je opisano u prethodnom pasusu). Oba rešavanja svođenjem na SAT su rešila svaku od generisanih instanci dimenzije 25×25 za najviše 1.6 sekunde. Pošto je u originalnom radu [83] navedeno da simulirano kaljenje nije rešilo više od 80% zagonetki dimenzije 25×25 za vremensko ograničenje od 350 sekunde po zagonetki (na jačem računaru), možemo zaključiti da je rešavanje svođenjem na SAT značajno efikasnije od pomenute heuristike pri rešavanju sudoku zagonetki.

Verovatnosni grafovski modeli su takođe upotrebljavani u rešavanju sudoku zagonetki [74]. Efikasnost ovog pristupa u originalnom radu je testirana na zagonetkama različitih stepena popunjenosti – pristup nije uspeo da reši neke zagonetke dimenzije 16×16. Iako nismo uspeli da dobijemo zagonetke koje su originalno korišćene (veb strana navedena u radu nije bila dostupna), rezultati u tabeli 4.5 su dati na zagonetkama vrlo raznolikih stepena popunjenosti. Pošto su oba rešavanja svođenjem na SAT rešila sve zagonetke dimenzije 16×16 u proseku za $61/400 = 0.15$ sekundi,

Tabela 4.5: Poređenje rešavanja originalnog svođenja problema Sudoku na SAT sa rešavanjem svođenjem na SAT koje koristi preprocesiranje. Vremensko ograničenje po zagonetki je 300 sekundi. U svakom polju su prikazani broj rešenih instanci i u zagradi ukupno utrošeno vreme u sekundama.

Dimenzija	Broj instanci	Originalno svođenje		Svođenje sa preproc.	
16×16	400	400	(61)	400	(61)
25×25	400	400	(163)	400	(158)
36×36	400	384	(7231)	383	(7216)

zaključujemo da je rešavanje svođenjem na SAT značajno efikasnije.

Takođe smo poredili efikasnost rešavanja svođenjem na SAT sa paralelnim rešavanjem sudoku zagonetki [34] na 20 zagonetki generisanih na način opisan u originalnom radu, pošto originalne zagonetke nisu javno dostupne. Paralelno rešavanje je rađeno na jačem računaru koji je koristio više jezgara istovremeno, a prosečno vreme na zagonetkama dimenzije 16×16 je mereno u sekundama. Oba rešavanja svođenjem na SAT su rešila svaku od 20 generisanih zagonetki i to u proseku za 0.11 sekundi, pa smatramo da su ovo značajno bolji rezultati u odnosu na paralelno rešavanje. Uzimajući u obzir da se rešavanje svođenjem na SAT izvršava samo na jednom a paralelno izvršavanje na više jezgara istovremeno, rezultati u korist rešavanja svođenjem na SAT su još ubedljiviji.

4.2.3.2 Eksperimenti koji se odnose na generisanje

Poredili smo originalni algoritam za generisanje velikih zagonetki sa našim unapređenim algoritmom (oba opisana u poglavlju 3). Naš algoritam smo koristili sa i bez preprocesiranja. Za svaki algoritam smo generisali po 100 zagonetki dimenzija 16×16 i 25×25. Prosečno vreme generisanja je prikazano u tabeli 4.6. Naš algoritam bilo bez ili sa preprocesiranjem pre svođenja na SAT se pokazao značajno efikasnijim od originalnog algoritma. Generisanje uz pomoć preprocesiranja se pokazalo efikasnijim od generisanja bez preprocesiranja. Generisanje pomoću bilo kog od tri algoritma za zagonetke dimenzije 36×36 nije bilo završeno ni posle više sati, pa nisu prikazani rezultati za ovu dimenziju. Potrebno je razviti efikasniji algoritam za generisanje zagonetki ove i većih dimenzija u budućnosti.

Tabela 4.7 prikazuje prosečne vrednosti nekih podataka vezanih za proces generisanja. Broj rešavanja svođenjem na SAT i samim tim i broj pokretanja SAT

Tabela 4.6: Poređenje originalnog algoritma za generisanje Sudoku zagonetki sa dve njegove modifikacije, jednom koja ne koristi preprocesiranje i drugom koja ga koristi. Za svaku dimenziju i svaki algoritam dato je prosečno vreme generisanja u sekundama na 100 zagonetki.

Dimenzija	Originalno gen.	Modif. gen.	Modif. gen. sa preproc.
16×16	845	17	13
25×25	2916	372	283

Tabela 4.7: Neki podaci generisanja Sudoku zagonetki dimenzija 16×16 / 25×25. Za svaku dimenziju date su prosečne vrednosti za 100 generisanja.

Karakteristika	Originalno gen.	Modif. gen.	Modif. gen. sa preproc.
Popunjenost zagonetke	36 / 43	35 / 41	35 / 41
Broj rešavanja svođenjem na SAT	257 / 626	135 / 319	127 / 311
Br. korišć. pravila *Jedna preostala vrednost*	-	-	402 / 343
Br. korišć. pravila *Jedno preostalo polje*	-	-	2973 / 9375
Br. korišć. pravila *Putna konzistencija*	-	-	26 / 7

rešavača je značajno smanjen u odnosu na originalni algoritam (skoro dvostruko). Razlog tome je da je posle prvog dela procesa generisanja procenat popunjenosti bio oko 50% – rad na tome da se ova zagonetka u drugom delu što je moguće više isprazni podrazumeva značajno manji broj zagonetki koje je potrebno rešiti, u poređenju sa pražnjenjem pune zagonetke kao u originalnom algoritmu. U slučaju kada se koristi preprocesiranje, broj rešavanja svođenjem na SAT je manji nego u slučaju kada se preprocesiranje ne koristi – razlog je što se samo pomoću preprocesiranja neke zagonetke u potpunosti rešavaju. Od korišćenih pravila preprocesiranja najviše je korišćeno pravilo Jedno preostalo polje dok je najmanje korišćeno pravilo Putna konzistencija. Ako bi se od navedena 3 pravila koristilo samo jedno, onda pravilo Jedno preostalo polje ima ubedljivo najveći uticaj na smanjenje vremena generisanja (ovo je potvrđeno eksperimentalnim rezultatima koji ovde nisu prikazani). Međutim, korišćenje sva 3 pravila ima prednost da izračunavanje vrednosti

nekog polja u zagonetki pomoću jednog od pravila povlači moguće izračunavanje vrednosti nekog drugog polja pomoću nekog drugog pravila. Na primer, ako bi se koristilo samo pravilo Jedna preostala vrednost pri rešavanju zagonetki dimenzija 25×25, onda bi broj primena tog pravila bio značajno smanjen, sa prosečnih 343 datih u tabeli 4.7 na prosečnih 164. Iz ove činjenice sledi da pri primeni pravila preprocesiranja treba koristiti kombinacije različitih pravila, a detaljnija procena efikasnosti različitih kombinacija jeste jedan mogući pravac daljeg rada.

Glava 5

Zaključci i dalji rad

5.1 Zaključci

U fokusu ove teze je rešavanje problema CSP svođenjem na problem SAT. Doprinosi se mogu podeliti u nekoliko kategorija.

Opisi savremenih tehnika rešavanja problema CSP. Jedan od ciljeva teze je bio metodičan opis postojećih tehnika rešavanja problema CSP. Predstavljeni su i jezici koji se koriste od strane raznoraznih CSP rešavača kao i sami rešavači. Pored opisa opštih, dati su i opisi tehnika svođenja problema CSP na SAT. Ove tehnike, koje se zovu i kodiranja, predstavljene su kroz primere.

Opis novih hibridnih kodiranja za rešavanje problema CSP. Predstavljena su i dva nova hibridna kodiranja za svođenje na SAT. Eksperimentima je potvrđeno poboljšanje u efikasnosti novih kodiranja u odnosu na kodiranja iz kojih su nastala. Za varijantu kodiranja uređenja dat je i dokaz ispravnosti tog kodiranja, koji do sada nije postojao u literaturi. Predstavljena je i analiza efikasnosti kodiranja na pojedinim tipovima problema i date su smernice za odabir kodiranja u zavisnosti od promenljivih i ograničenja koje se javljaju.

Sistem *meSAT*. Razvijen je sistem *meSAT* koji podržava 4 osnovna i 2 hibridna kodiranja pri svođenju na SAT. Sistem omogućava i korišćenje različitih implementacija bulovskih ograničenja kardinalnosti. *meSAT* omogućava rešavanje problema svođenjem i na SMT i PB, probleme bliske problemu SAT. Razvijeni sistem omo-

gućava rešavanje problema pomoću velikog broja različitih metoda, što ga čini pogodnim za evaluaciju različitih metoda.

Portfolio za odabir kodiranja/rešavača CSP instance. Opisane su postojeće tehnike mašinskog učenja kao i portfoliji koji su razvijeni za rešavanje problema SAT i CSP. Razvijen je i portfolio ArgoCSP-kNN koji je zasnovan na postojećem SAT portfoliju ArgoSmArT, ali je prilagođen CSP instancama. Eksperimenti su pokazali da se portfolio pokazao uspešnim u automatskom odabiru kodiranja pri svođenju na SAT. Sprovedeni su i obimni eksperimenti koji se odnose na odabir CSP rešavača pomoću ovog portfolija. Rezultati pokazuju da je ArgoCSP-kNN uporediv sa vodećim savremenim portfolijima.

Primena portfolija zasnovana na kratkom treniranju. Napravljena je procena uticaja dužine treniranja na kvalitet odabira rešavača. Predstavljen je prikaz zasnovan na kratkom treniranju i pokazano je da je on konkurentan postojećim pristupima koji koriste znatno duže treniranje. Izvršeno je i poređenje nekoliko tehnika mašinskog učenja, sa ciljem utvrđivanja koje tehnike su pogodnije za pristup zasnovan na kratkom treniranju.

Rešavanje problema raspoređivanja kontrolora leta. Problem raspoređivanja kontrolora leta je detaljno prikazan, kao i tri modela ovog problema. Problem je rešavan velikim brojem različitih metoda i rešavača. Razvijena je metoda koja predstavlja kombinaciju rešavanja problema svođenjem na SAT i korišćenjem lokalne pretrage. Ova metoda se u sprovedenim eksperimentima pokazuje kao najefikasnija za rešavanje problema raspoređivanja kontrolora.

Generisanje i rešavanje velikih sudoku zagonetki svođenjem na SAT. Sprovedeni su testovi sa ciljem da se uporedi najpoznatije postojeće rešavanje velikih sudoku zagonetki svođenjem na SAT sa ostalim metodama. Pokazano je da je svođenje na SAT značajno efikasnije od ostalih metoda. Uvedena su vrlo jednostavna pravila preprocesiranja i unapređena je efikasnost postojećeg algoritma za generisanje velikih sudoku zagonetki. Efikasnost generisanja se dodatno se unapređuje korišćenjem uvedenih pravila preprocesiranja.

5.2 Dalji rad

Primena portfolija zasnovana na kratkom treniranju. Interesantno bi bilo videti kako se pristup sa kratkim treniranjem prenosi i na druge domene (ASP, MaxSAT, itd.) pa je ovo jedan od mogućih pravaca budućeg rada. Dodatna istraživanja su potrebna i da bi se utvrdilo uopšte pod kojim uslovima može da se primeni kratko treniranje. Razvijanje metodologije kojom bi se automatski utvrđivalo koje vremenskog ograničenje dati rešavačima tokom kratkog treniranja je interesantan, ali i veoma zahtevan zadatak.

Primene CSP rešavača. Iako su zbog veoma velikog broja tvrdih ograničenja metaheuristike manje pogodne za rešavanje problema raspoređivanja kontrolora nego za neke druge probleme, interesantno bi bilo detaljno ispitati da li i koje metaheuristike su pogodne za taj problem. Zanimljiv pravac predstavlja i primena portfolija ArgoCSP-kNN na problem raspoređivanja kontrolora leta, sa ciljem poboljšanja efikasnosti u rešavanju ovog problema.

Generisanje i rešavanje velikih sudoku zagonetki svođenjem na SAT. Kod sudoku problema bi trebalo utvrditi koje su tehnike preprocesiranja pogodne za veće zagonetke. Ove tehnike bi mogle dodatno da ubrzaju rešavanje i generisanje zagonetki kao i da omoguće i rešavanje zagonetki većih dimenzija.

Bibliografija

[1] Ignasi Abío, Valentin Mayer-Eichberger, and Peter J. Stuckey. Encoding linear constraints with implication chains to CNF. In *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015*, volume 9255 of *LNCS*, pages 3–11. Springer, 2015.

[2] Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. An empirical evaluation of portfolios approaches for solving csps. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 10th International Conference, CPAIOR 2013*, volume 7874 of *LNCS*, pages 316–324. Springer, 2013.

[3] Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. SUNNY: a lazy portfolio approach for constraint solving. *Theory and Practice of Logic Programming*, 14(4-5):509–524, 2014.

[4] Krzysztof R. Apt. *Principles of constraint programming*. Cambridge University Press, 2003.

[5] Krzysztof R Apt and Mark Wallace. *Constraint logic programming using ECLiPSe*. Cambridge University Press, 2006.

[6] M Arnvig, B Beermann, B Köper, M Maziul, U Mellett, C Niesing, and J Vogt. Managing shiftwork in european atm. *Literature Review. European Organisation for the safety of air navigation*, 2006.

[7] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks and their applications. In *Theory and Applications of Satisfiability Testing - SAT 2009*, volume 5584 of *LNCS*, pages 167–180. Springer, 2009.

[8] Thanasis Balafoutis, Anastasia Paparrizou, and Kostas Stergiou. Experimental evaluation of branching schemes for the csp. *arXiv preprint arXiv:1009.0407*, 2010.

[9] Mutsunori Banbara, Haruki Matsunaka, Naoyuki Tamura, and Katsumi Inoue. Generating combinatorial test cases by efficient sat encodings suitable for cdcl sat solvers. In *Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17*, volume 6397 of *LNCS*, pages 112–126. Springer, 2010.

[10] Milan Banković. Extending smt solvers with support for finite domain alldifferent constraint. *Constraints*, pages 1–32.

[11] Roman Bartak. Constraint propagation and backtracking-based search. *Charles Universität, Prag*, 2005.

[12] Nicolas Beldiceanu, Mats Carlsson, and Jean-Xavier Rampon. Global constraint catalog. Technical report, SICS, 2005.

[13] Nicolas Beldiceanu and Evelyne Contejean. Introducing global constraints in chip. *Mathematical and computer Modelling*, 20(12):97–123, 1994.

[14] Yael Ben-Haim, Alexander Ivrii, Oded Margalit, and Arie Matsliah. Perfect hashing and cnf encodings of cardinality constraints. In *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference*, volume 7317 of *LNCS*, pages 397–409. Springer, 2012.

[15] Bo Bernhardsson. Explicit solutions to the n-queens problem for all n. *ACM SIGART Bulletin*, 2(2):7, 1991.

[16] Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–6, 2010.

[17] Christian Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65(1):179–190, 1994.

[18] Armin Biere. Lingeling, plingeling, picosat and precosat at sat race 2010. *FMV Report Series Technical Report*, 10(1), 2010.

[19] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[20] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Thomas Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger H. Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016.

[21] Christopher M Bishop et al. *Pattern recognition and machine learning*, volume 4. Springer New York, 2006.

[22] Gustav Björdal, Jean-Noël Monette, Pierre Flener, and Justin Pearson. A constraint-based local search backend for minizinc. *Constraints*, 20(3):325–345, 2015.

[23] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[24] Miquel Bofill, Josep Suy, and Mateu Villaret. A system for solving constraint satisfaction problems with smt. In *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010*, volume 6175 of *LNCS*, pages 300–305. Springer, 2010.

[25] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004*, pages 146–150. IOS Press, 2004.

[26] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[27] Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–499, 2004.

[28] Marco Cadoli, Luigi Palopoli, Andrea Schaerf, and Domenico Vasile. NP-SPEC: An executable specification language for solving all problems in np. In *Practical Aspects of Declarative Languages, First International Workshop, PADL '99*, volume 1551 of *LNCS*, pages 16–30. Springer, 1999.

[29] Mirjana Čangalović, Vera Kovačević-Vujčić, Lav Ivanović, and Milan Dražić. Modeling and solving a real-life assignment problem at universities. *European Journal of Operational Research*, 110(2):223–233, 1998.

[30] Alberto Caprara, Paolo Toth, and Matteo Fischetti. Algorithms for the set covering problem. *Annals OR*, 98(1-4):353–371, 2000.

[31] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27, 2011.

[32] Jingchao Chen. A new sat encoding of the at-most-one constraint. In *Proceedings of the 9th International Workshop on Constraint Modelling and Reformulation*, 2010.

[33] Marco Chiarandini, Mauro Birattari, Krzysztof Socha, and Olivia Rossi-Doria. An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, 9(5):403–432, 2006.

[34] Alton Chiu, Ehsan Nasiri, and Rafat Rashid. Parallelization of sudoku. *University of Toronto, Technical Report*, 2012.

[35] Michael Codish, Yoav Fekete, Carsten Fuhs, and Peter Schneider-Kamp. Optimal base encodings for pseudo-boolean constraints. In *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011*, volume 6605 of *LNCS*, pages 189–204. Springer, 2011.

[36] Michael Codish and Moshe Zazon-Ivry. Pairwise cardinality networks. In *Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17*, volume 6355 of *LNCS*, pages 154–172. Springer, 2010.

[37] Stephen A. Cook. Proceedings of the 3rd annual acm symposium on theory of computing. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971.

[38] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[39] Dragoš Cvetković, Mirjana Čangalović, Đorđe Dugošija, Vera Kovačević-Vujčić, Slobodan Simić, Jovo Vuleta, and Dragoš Cvetković. *Kombinatorna optimizacija*. Društvo operacionih istraživača Jugoslavije, 1996.

[40] Johan de Kleer. A comparison of ATMS and CSP techniques. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 290–296. Morgan Kaufmann, 1989.

[41] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[42] Rina Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.

[43] Rina Dechter and Judea Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1987.

[44] Bruno Dutertre and Leonardo de Moura. The Yices SMT solver. Technical report, SRI International, 2006.

[45] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.

[46] Niklas Eén and Niklas Sörensson. Temporal induction by incremental sat solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003.

[47] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006.

[48] EUROCONTROL. Shiftwork practices study - atm and related industries. *DAP/SAF-2006/56 Brussels : EUROCONTROL*, 2006.

[49] Yoav Fekete and Michael Codish. Simplifying pseudo-boolean constraints in residual number systems. In *Theory and Applications of Satisfiability Testing–SAT 2014*, LNCS, pages 351–366. Springer, 2014.

[50] Thibaut Feydy and Peter J. Stuckey. Lazy clause generation reengineered. In *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009*, volume 5732 of *LNCS*, pages 352–366. Springer, 2009.

[51] Committee for a Review of the En Route Air Traffic Control Complexity and Workload Model. Air traffic controller staffing in the en route domain: A review of the federal aviation administration's task load model. 2010.

[52] Robert Fourer and David M. Gay. Extending an algebraic modeling language to support constraint programming. *INFORMS Journal on Computing*, 14(4):322–344, 2002.

[53] Marco Gavanelli. The log-support encoding of csp into sat. In *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007*, volume 4741 of *LNCS*, pages 815–822. Springer, 2007.

[54] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. *clasp* : A conflict-driven answer set solver. In *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007*, volume 4483 of *LNCS*, pages 260–265. Springer, 2007.

[55] Allen Van Gelder. Another look at graph coloring via propositional satisfiability. *Discrete Applied Mathematics*, 156(2):230–243, 2008.

[56] Ian P. Gent. Arc consistency in sat. In *Proceedings of the 15th Eureopean Conference on Artificial Intelligence, ECAI'2002*, pages 121–125. IOS Press, 2002.

[57] Ian P Gent and Toby Walsh. Csplib: a benchmark library for constraints. In *Principles and Practice of Constraint Programming–CP'99*, pages 480–481. Springer, 1999.

[58] Matthew L. Ginsberg, Michael Frank, Michael P. Halpin, and Mark C. Torrance. Search lessons learned from crossword puzzles. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 210–215. AAAI Press / The MIT Press, 1990.

[59] Liana Hadarean, Kshitij Bansal, Dejan Jovanovic, Clark Barrett, and Cesare Tinelli. A tale of two solvers: Eager and lazy approaches to bit-vectors. In *Computer Aided Verification - 26th International Conference, CAV 2014*, volume 8559 of *LNCS*, pages 680–695. Springer, 2014.

[60] Emmanuel Hebrard. Mistral, a constraint satisfaction library. *Proceedings of the 3rd International CSP Solver Competition*, pages 31–39, 2008.

[61] Pascal Van Hentenryck, Laurent Michel, Laurent Perron, and Jean-Charles Régin. Constraint programming in OPL. In *Principles and Practice of Declarative Programming, International Conference PPDP'99*, volume 1702 of *LNCS*, pages 98–116. Springer, 1999.

[62] Jinbo Huang. Universal booleanization of constraint models. In *Principles and Practice of Constraint Programming, 14th International Conference, CP 2008*, volume 5202 of *LNCS*, pages 144–158. Springer, 2008.

[63] Barry Hurley, Lars Kotthoff, Yuri Malitsky, and Barry O'Sullivan. Proteus: A hierarchical portfolio of solvers and transformations. In *Integration of AI and OR Techniques in Constraint Programming - 11th International Conference, CPAIOR 2014*, volume 8451 of *LNCS*, pages 301–317. Springer, 2014.

[64] Frank Hutter, Domagoj Babic, Holger H. Hoos, and Alan J. Hu. Boosting verification by automatic tuning of decision procedures. In *Formal Methods in Computer-Aided Design, 7th International Conference, FMCAD 2007*, pages 27–34. IEEE Computer Society, 2007.

[65] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[66] Frank Hutter, Manuel López-Ibáñez, Chris Fawcett, Marius Thomas Lindauer, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. Aclib: A benchmark library for algorithm configuration. In *Learning and Intelligent Optimization - 8th International Conference*, volume 8426 of *LNCS*, pages 36–40. Springer, 2014.

[67] Joey Hwang and David G. Mitchell. 2-way vs. d-way branching for CSP. In *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005*, volume 3709 of *LNCS*, pages 343–357. Springer, 2005.

[68] Kazuo Iwama and Shuichi Miyazaki. Sat-varible complexity of hard combinatorial problems. In *Technology and Foundations - Information Processing '94, Volume 1, Proceedings of the IFIP 13th World Computer Congress*, pages 253–258, 1994.

[69] Predrag Janičić. Uniform reduction to sat. *Logical Methods in Computer Science*, 8(3), 2010.

[70] Narendra Jussien, Guillaume Rochart, and Xavier Lorca. Choco: an open source java constraint programming library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, pages 1–10, 2008.

[71] Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011*, volume 6876 of *LNCS*, pages 454–469. Springer, 2011.

[72] Muhammad Rezaul Karim. A new approach to constraint weight learning for variable ordering in csps. *CoRR*, abs/1312.6996, 2013.

[73] Simon Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, 45(3):275–286, 1990.

[74] Sheehan Khan, Shahab Jabbari, Shahin Jabbari, and Majid Ghanbarinejad. Solving sudoku using probabilistic graphical models. 2014.

[75] Zeynep Kiziltan, Luca Mandrioli, Barry O'Sullivan, and Jacopo Mauro. A classification-based approach to manage a solver portfolio for csps. In *Proceedings of the 22nd Irish Conference on Artificial Intelligence and Cognitive Science, AICS-2011*, 2011.

[76] Will Klieber and Gihwon Kwon. Efficient cnf encoding for selecting 1 from n objects. In *Proc. International Workshop on Constraints in Formal Verification*, 2007.

[77] Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. Qmaxsat: A partial max-sat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(1/2):95–100, 2012.

[78] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60, 2014.

[79] Sava Krstić and Amit Goel. Architecting solvers for SAT modulo theories: Nelson-oppen with DPLL. In *Frontiers of Combining Systems, 6th Internati-*

onal Symposium, FroCoS 2007, volume 4720 of *LNCS*, pages 1–27. Springer, 2007.

[80] Gihwon Kwon and Himanshu Jain. Optimized cnf encoding for sudoku puzzles. In *Proc. 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR2006)*, pages 1–5, 2006.

[81] Rhyd Lewis. Metaheuristics can solve sudoku puzzles. *Journal of Heuristics*, 13(4):387–401, 2007.

[82] Inês Lynce and Joël Ouaknine. Sudoku as a sat problem. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM 2006)*, 2006.

[83] Marlos C. Machado and Luiz Chaimowicz. Combining metaheuristics and csp algorithms to solve sudoku. In *SBGames*, pages 124–131. IEEE, 2011.

[84] Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[85] Yuri Malitsky, Deepak Mehta, and Barry O'Sullivan. Evolving instance specific algorithm configuration. In *Proceedings of the Sixth Annual Symposium on Combinatorial Search, SOCS 2013*. AAAI Press, 2013.

[86] Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Non-model-based algorithm portfolios for sat. In *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011*, volume 6695 of *LNCS*, pages 369–370. Springer, 2011.

[87] Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Satisfiability solver selector (3s). *SAT CHALLENGE 2012*, page 50, 2012.

[88] Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm portfolios based on cost-sensitive hierarchical clustering. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 608–614. AAAI Press, 2013.

[89] Yuri Malitsky and Meinolf Sellmann. Instance-specific algorithm configuration as a method for non-model-based portfolio generation. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems - 9th International Conference, CPAIOR 2012*, volume 7298 of *LNCS*, pages 244–259. Springer, 2012.

[90] Filip Marić. Formalization and implementation of modern SAT solvers. *Journal of Automated Reasoning*, 43(1):81–119, 2009.

[91] Filip Marić and Predrag Janičić. Urbiva: Uniform reduction to bit-vector arithmetic. In *Automated Reasoning, 5th International Joint Conference, IJCAR 2010*, volume 6173 of *LNCS*, pages 346–352. Springer, 2010.

[92] Joao Marques-Silva. The msuncore maxsat solver. *SAT 2009 competitive events booklet: preliminary version*, page 151, 2009.

[93] Sylvain Merchez, Christophe Lecoutre, and Frédéric Boussemart. Abscon: A prototype to solve csps with abstraction. In *Principles and Practice of Constraint Programming - CP 2001, 7th International Conference, CP 2001*, volume 2239 of *LNCS*, pages 730–744. Springer, 2001.

[94] Amit Metodi and Michael Codish. Compiling finite domain constraints to sat with bee. *Theory and Practice of Logic Programming*, 12(4-5):465–483, 2012.

[95] Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

[96] Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information sciences*, 7:95–132, 1974.

[97] Malek Mouhoub and Bahareh Jafari Jashmi. Heuristic techniques for variable and value ordering in csps. In *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011*, pages 457–464. ACM, 2011.

[98] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007*, volume 4741 of *LNCS*, pages 529–543. Springer, 2007.

[99] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll(T). *Journal of the ACM*, 53(6):937–977, 2006.

[100] Mladen Nikolić. Statistical methodology for comparison of sat solvers. In *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010*, volume 6175 of *LNCS*, pages 209–222. Springer, 2010.

[101] Mladen Nikolić, Filip Marić, and Predrag Janičić. Instance-based selection of policies for sat solvers. In *Theory and Applications of Satisfiability Testing - SAT 2009*, volume 5584 of *LNCS*, pages 326–340. Springer, 2009.

[102] Mladen Nikolić, Filip Marić, and Predrag Janičić. Simple algorithm portfolio for sat. *Artificial Intelligence Review*, pages 1–9, 2011.

[103] Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.

[104] Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science, AICS-2008*, 2008.

[105] Laurent Perron. Operations research and constraint programming at google. In *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011*, volume 6876 of *LNCS*, page 2. Springer, 2011.

[106] Jean-Charles Régin. Global constraints and filtering algorithms. In *Constraint and Integer Programming*, pages 89–135. Springer, 2004.

[107] John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[108] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.

[109] Olivier Roussel and Christophe Lecoutre. Xml representation of constraint networks: Format xcsp 2.1. *CoRR*, abs/0902.2362, 2009.

[110] Christian Schulte, Mikael Lagerkvist, and Guido Tack. Gecode. *Software download and online material at the website: http://www. gecode. org*, 2006.

[111] Mirko Vujošević. *Metode Optimizacije u Inženjerskom Menadžmentu*. Akademija inženjerskih nauka Srbije, 2012.

[112] Mirko Vujošević. Programiranje ograničenja. *Info M*, 44:4–10, 2012.

[113] Bryan Silverthorn and Risto Miikkulainen. Latent class models for algorithm portfolio methods. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*. AAAI Press, 2010.

[114] Helmut Simonis. Sudoku as a constraint problem. In *CP Workshop on modeling and reformulating Constraint Satisfaction Problems*, volume 12, pages 13–27. Citeseer, 2005.

[115] Carsten Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. In *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005*, volume 3709 of *LNCS*, pages 827–831. Springer, 2005.

[116] Alex Smola and Vladimir Vapnik. Support vector regression machines. *Advances in neural information processing systems*, 9:155–161, 1997.

[117] Gert Smolka. The oz programming model. In *Logics in Artificial Intelligence, European Workshop, JELIA '96*, volume 1126 of *LNCS*, page 251. Springer, 1996.

[118] Takehide Soh, Katsumi Inoue, Naoyuki Tamura, Mutsunori Banbara, and Hidetomo Nabeshima. A sat-based method for solving the two-dimensional strip packing problem. *Fundamenta Informaticae*, 102(3-4):467–487, 2010.

[119] Peter Steinke and Norbert Manthey. Pblib—a c++ toolkit for encoding pseudo–boolean constraints into cnf. *TU Dresden, Dresden, Germany, Technical Report*, 1:2014, 2014.

[120] Mirko Stojadinović. Air traffic controller shift scheduling by reduction to csp, SAT and sat-related problems. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014*, volume 8656 of *LNCS*, pages 886–902. Springer, 2014.

[121] Mirko Stojadinović. Generisanje i rešavanje velikih sudoku zagonetki svođenjem na sat problem. *Info M*, 54:25–30, 2015.

[122] Mirko Stojadinović and Filip Marić. mesat: multiple encodings of CSP to SAT. *Constraints*, 19(4):380–403, 2014.

[123] Peter J Stuckey, Thibaut Feydy, Andreas Schutt, Guido Tack, and Julien Fischer. The minizinc challenge 2008–2013. *AI Magazine*, 35(2):55–60, 2014.

[124] Naoyuki Tamura and Mutsunori Banbara. Sugar: A csp to sat translator based on order encoding. In *Proceedings of the third constraint solver competition*, pages 65–69, 2008.

[125] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear csp into sat. *Constraints*, 14(2):254–272, 2009.

[126] Tomoya Tanjo, Naoyuki Tamura, and Mutsunori Banbara. Sugar++: a sat-based max-csp/cop solver. *Proc. the Third International CSP Solver Competition*, pages 144–151, 2008.

[127] Tomoya Tanjo, Naoyuki Tamura, and Mutsunori Banbara. A compact and efficient sat-encoding of finite domain csp. In *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011*, volume 6695 of *LNCS*, pages 375–376. Springer, 2011.

[128] Tomoya Tanjo, Naoyuki Tamura, and Mutsunori Banbara. Azucar: A sat-based csp solver using compact order encoding - (tool presentation). In *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference*, volume 7317 of *LNCS*, pages 456–462. Springer, 2012.

[129] EATCHIP Human Resources Team. Ats manpower planning in practice: Introduction to a qualitative and quantitative staffing methodology. *HUM.ET1.ST02.2000-REP-01 Brussels : EUROCONTROL*, 1998.

[130] Andrija Tomović, Predrag Janičić, and Vlado Kešelj. n-gram-based classification and unsupervised hierarchical clustering of genome sequences. *Computer Methods and Programs in Biomedicine*, 81(2):137–153, 2006.

[131] G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, pages 466–483. Springer, Berlin, Heidelberg, 1983.

[132] Toby Walsh. Sat v csp. In *Principles and Practice of Constraint Programming - CP 2000, 6th International Conference*, volume 1894 of *LNCS*, pages 441–456. Springer, 2000.

[133] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. Swi-prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012.

[134] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

[135] Günther Zäpfel and Roland Braune. *Metaheuristic search concepts: A tutorial with applications to production and logistics*. Springer Science & Business Media, 2010.

[136] Neng-Fa Zhou. The language features and architecture of b-prolog. *Theory and Practice of Logic Programming*, 12(1-2):189–218, 2012.

Biografija autora

Mirko Stojadinović rođen je u Smederevskoj Palanci 28. oktobra 1985. godine. Osnovnu školu "Heroj Ivan Muker" i gimnaziju "Sveta Đorđević" je obe završio sa prosečnom ocenom 5.00 i kao nosilac Vukovih diploma. Proglašen je za đaka generacije gimnazije. Učestvovao je na velikom broju takmičenja iz matematike, srpskog i engleskog jezika. Najvažnije nagrade koje je osvojio su dva druga mesta iz matematike na republičkim smotrama mladih talenata koje su održane u Kladovu.

Studije na smeru Računarstvo i informatika na Matematičkom fakultetu Univerziteta u Beogradu upisao je školske 2004/05 godine, a diplomirao je u julu 2009. godine sa prosečnom ocenom 9.53. Tokom studija je učestvovao u radu studentskog parlamenta, a bio je aktivan i kao član studentske organizacije AIESEC. U okviru organizacije IAESTE je išao na dvomesečnu stručnu praksu u Lođ, Poljska, gde je radio na razvoju sistemskih alatki za operativni sistem QNX. Kao jedan od 200 najboljih studenata, putovao je Evropom u organizaciji Evropskog pokreta u Srbiji. Bio je stipendista Republičke fondacije za razvoj naučnog i umetničkog podmlatka.

Doktorske akademske studije na smeru Informatika upisao je u oktobru 2009. godine. Tokom studija je imao izlaganja na nekoliko konferencija i radionica u zemlji i inostranstvu. Od oktobra 2009. bio je zaposlen je kao saradnik u nastavi, a od oktobra 2011. do januara 2016. godine kao asistent na Katedri za računarstvo i informatiku Matematičkog fakulteta Univerziteta u Beogradu. Tokom dosadašnjeg rada na Matematičkom fakultetu držao je vežbe iz 6 predmeta. Od 2011. do 2016. godine bio je uključen u projekat 174021 Ministarstva prosvete i nauke Republike Srbije, a tokom tog perioda bio je i član grupe za automatsko rezonovanje Matematičkog fakulteta. Od januara 2016. godine zaposlen je kao softverski inženjer u kompaniji Microsoft Development Center Serbia.

Englesko-srpski rečnik

air traffic controller (ATCo)	kontrolor leta[1]
algorithm configuration problem	problem konfigurisanja algoritma
algorithm portfolio	algoritamski portfolio
algorithm selection problem	problem izbora algoritma
arc consistency	lučna konzistencija
artficial neural networks	veštačke neuronske mreže
ATCo Shift Scheduling Problem	problem raspoređivanja kontrolora leta
backjump clause	klauza povratnog skoka
backjumping	pretraga sa skokom unazad
backmarking	pretraga sa obeležavanjem
backtracking	pretraga sa povratkom
boolean cardinality constraint	bulovsko ograničenje kardinalnosti
boolean satisfiability problem (SAT)	problem iskazne zadovoljivosti
branch and bound	grananje i ograničavanje
branching	grananje
branching strategy	strategija grananja
chronological backtracking	hronološka pretraga sa povratkom
classification	klasifikacija
clause	klauza
clause learning	učenje klauza
conflict analysis	analiza konflikta
conflict-driven backjumping	pretraga sa skokom unazad vođena analizom konflikta
conjunctive normal form (CNF)	konjunktivna normalna forma (KNF)
consistent	konzistentan

[1] Ovo je rečnik pojmova korišcen u tezi. Najveći broj pojmova može se prevesti i na druge načine.

constraint logic programming	logičko programiranje uz ograničenja
constraint optimization problem (COP)	problem optimizacije uz ograničenja
constraint programming (CP)	programiranje ograničenja
constraint propagation	propagiranje ograničenja
constraint satisfaction problem (CSP)	problem zadovoljenja ograničenja
constraint solver	rešavač ograničenja
course timetabling	pravljenje rasporeda časova
cross validation	unakrsna provera
decision literal	pretpostavljeni literal
direct encoding	direktno kodiranje
direct-order encoding	direktno kodiranje uređenja
eager	vredan
empirical hardness model	empirijski model težine
enumeration	potpuno nabrajanje
equisatisfiable	ekvizadovoljiv
extensional constraint	ekstenzionalno ograničenje
feature	atribut
first unique implication point	prva tačka jedinstvene implikacije
fold	deo
forward checking	proveravanje unapred
full look ahead	potpuni pogled unapred
global constraint	globalno ograničenje
hard clause	tvrda klauza
hard constraint	tvrdo ograničenje
hill climbing	penjanje-uz-brdo
implied literal	izvedeni literal
intensional constraint	intenzionalno ograničenje
iterative improvement	iterativno poboljšavanje
k-nearest neighbors	k-najbližih suseda
lazy	lenji
lazy clause generation	lenjo generisanje klauza
learned clause	naučena klauza
literal	literal
local search	lokalna pretraga
log encoding	logaritamsko kodiranje

machine learning (ML)	mašinsko učenje
maintaining arc consistency (MAC)	održavanje lučne konzistencije
metaheuristics	metaheuristike
min-conflict heuristic	heuristika minimizacije konflikta
modeling	modeliranje
node consistency	čvorna konzistencija
nurse scheduling problem	problem raspoređivanja medicinskih sestara
objective function	funkcija cilja
opposite literal	suprotan literal
order encoding	kodiranje uređenja
partial look ahead	delimičan pogled unapred
path consistency	putna konzistencija
propagation	propagiranje
pseudo-boolean (PB)	pseudo-bulovsko
random forests	slučajne šume
regression	regresija
rest day	dan relaksacije
restarting	otpočinjanje iznova
ridge	ridž
SAT encoding	kodiranje na SAT
SAT solver	SAT rešavač
satisfiable	zadovoljiv
set covering problem	problem pokrivanja skupa
SMT solver	SMT rešavač
soft clause	meka klauza
soft constraint	meko ograničenje
stochastic local search methods	metode stohastičke lokalne pretrage
support encoding	kodiranje podrške
support vector machine	metoda podržavajućih vektora
support vector regression	regresija podržavajućih vektora
time limit	vremensko ograničenje
true/false	tačno/netačno
two-watch literal scheme	shema dva posmatrana literala
unit propagation	propagiranje jediničnih klauza
vacation day	dan odmora

working hour	radni sat
working shift	smena

Прилог 1.

Изјава о ауторству

Потписани-а _____

број индекса _____

Изјављујем

да је докторска дисертација под насловом

_____

_____

- резултат сопственог истраживачког рада,
- да предложена дисертација у целини ни у деловима није била предложена за добијање било које дипломе према студијским програмима других високошколских установа,
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио интелектуалну својину других лица.

Потпис докторанда

У Београду, _____

_____

Прилог 2.

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора _____

Број индекса _____

Студијски програм _____

Наслов рада _____

Ментор _____

Потписани/а _____

Изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао/ла за објављивање на порталу **Дигиталног репозиторијума Универзитета у Београду.**

Дозвољавам да се објаве моји лични подаци везани за добијање академског звања доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

Потпис докторанда

У Београду, _____

_____

Прилог 3.

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић“ да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

_____

_____

која је моје ауторско дело.

Дисертацију са свим прилозима предао/ла сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигитални репозиторијум Универзитета у Београду могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

1. Ауторство

2. Ауторство - некомерцијално

3. Ауторство – некомерцијално – без прераде

4. Ауторство – некомерцијално – делити под истим условима

5. Ауторство – без прераде

6. Ауторство – делити под истим условима

(Молимо да заокружите само једну од шест понуђених лиценци, кратак опис лиценци дат је на полеђини листа).

Потпис докторанда

У Београду, _____

_____