AUTOMATSKO REŠAVANJE IGRE SOKOBAN POMOĆU VEŠTAČKE INTELIGENCIJE

AUTOMATED SOLVING OF SOKOBAN PUZZLE USING ARTIFICIAL INTELLIGENCE

Milan Banković[1]
Matematički fakultet, Beograd[1]

Sadržaj – *Igra Sokoban je pomerajuća slagalica u kojoj igrač, čuvar skladišta, pomera kutije po tabli za igru, u cilju njihovog postavljanja na unapred zadata mesta. Pored toga što je zanimljiva za ručno rešavanje, ova igra inspiriše mnoge istraživače da se bave efikasnom automatizacijom njenog rešavanja, s obzirom na izuzetno veliku algoritamsku složenost ovog problema. U ovom radu, prikazujemo novi pristup zasnovan na automatskom planiranju uz pomoć procedura za ispitivanje zadovoljivosti u odnosu na teorije prvog reda, poznatih i kao SMT rešavači.*

Abstract – *The game of Sokoban is a sliding puzzle where a player, a warehouse keeper, pushes a number of boxes over the game board, in order to place them on the particular fields, given in advance. Besides the interest for manual solving, the game also attracts a great attention of the researchers to work on efficent automatization of its solving, because of the great algorithm complexity of the problem. In this paper, we propose a novel approach based on automated planning using procedures for checking satisfiability modulo first order theories, known as SMT solvers.*

1 UVOD

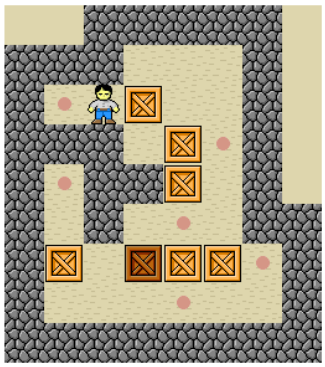
Mnogi problemi za čije je rešavanje ranije bio neophodan ljudski napor u današnje vreme se mogu uspešno rešavati primenom metoda koje spadaju u oblast veštačke inteligencije. Jedna zanimljiva klasa problema koji se mogu rešavati alatima zasnovanim na veštačkoj inteligenciji jesu i različite igre u kojima je potrebno pronaći strategiju koja igrača vodi do ispunjenja zadatog cilja. Poseban tip igara predstavljaju tzv. *slagalice* (engl. *puzzle*), u kojima igrač pokušava da pronađe rešenje (ili da konstruiše put do rešenja) u skladu sa zadatim pravilima. Slagalice su pre svega namenjene zabavi, kao i testiranju čovekove inteligencije, ali mogu biti veoma zanimljive i sa istraživačkog aspekta, jer su često u pitanju problemi (ili instance problema) čije je rešavanje u algoritamskom smislu izuzetno izazovno [1]. Otuda se pronalaženje efikasnih metoda za njihovo automatsko rešavanje može smatrati značajnim doprinosom u oblasti računarstva, jer se razvijeni metodi često mogu uopštiti i na druge probleme sa značajnijim primenama u praksi.

Postoji više opštih pristupa u veštačkoj inteligenciji među kojima u današnje vreme dominira *statistički pristup*, na kome je, u velikoj meri, zasnovan i danas najpopularniji pravac u veštačkoj inteligenciji poznat i kao *mašinsko učenje* (engl. *machine learning* [2, 3]). Nasuprot statističkom pristupu, *simbolički pristup* [4] se zasniva na formalnoj i preciznoj reprezentaciji problema, kao i rešavanju problema koristeći logičke metode i metode pretrage. Za razliku od statističkih metoda koji su po pravilu induktivne prirode, simbolički metodi su obično zasnovani na dedukciji, tj. na primeni rigoroznih pravila nad simboličkom reprezentacijom problema, u cilju pronalaženja rešenja koje zadovoljava zadate uslove.

Jedna od podoblasti veštačke inteligencije zasnovana na simboličkom pristupu je i *automatsko rezonovanje* (engl. *automated reasoning* [5]), koje se bavi automatizacijom procesa deduktivnog rasuđivanja. Automatsko rezonovanje se u najvećoj meri oslanja na formalne sisteme razvijene u okviru matematičke logike, uz primenu znanja iz drugih oblasti računarstva (poput teorijskog računarstva, algoritmike i sl.) i matematike (poput algebre, matematičke optimizacije i sl.). Glavna primena automatskog rezonovanja je u automatskom dokazivanju teorema, formalizaciji matematičkog znanja, kao i u razvoju i primeni formalnih metoda za verifikaciju hardvera, softvera, ali i drugih formalno zadatih sistema, poput kriptografskih i drugih protokola. Za razliku od statistički zasnovanih metoda, metode automatskog rezonovanja zahtevaju preciznu formulaciju problema, kao i pravila i uslova koje mora ispunjavati traženo rešenje. Takođe, rešenje koje dobijamo je potpuno precizno opisano i tačno sa veoma visokim stepenom pouzdanosti. Otuda su metode zasnovane na automatskom rezonovanju pogodne za probleme gde postoji nulta tolerancija na greške (npr. tamo gde mogu biti ugroženi ljudski životi, imovina i sl.).

U ovom radu razmatramo primenu metoda automatskog rezonovanja na rešavanje jedne od veoma popularnihih slagalica današnjice – igre *Sokoban* [6]. U pitanju je igra nastala 1981. godine, a kreirao ju je japanac Hirojuki Imbajaši. Igra se odvija na dvodimenzionoj tabli, poput one prikazane na slici 1. Igrač upravlja figurom koja predstavlja čuvara skladišta. Čuvar skladišta u svakom koraku može da se pomeri za jedno polje gore, dole, levo ili desno. Pritom, igrač ne može prolaziti kroz zidove. Igrač prilikom pomeranja može pomeriti (u smeru svog kretanja, tj. „pogurati") i neku od kutija koja mu se nađe na putu, ali samo pod uslovom da iza te kutije (u smeru pomeranja) nije zid ili druga kutija. Igrač ne može „povući" kutiju za sobom, već može samo

„gurati" kutije. Cilj igre je da se kutije postave na unapred zadata polja na tabli (na slici 1 označena crvenim tačkama). Pritom, nije bitno koja će kutija biti na kojoj odredišnoj poziciji. Cilj igre može biti pronaći bilo koje rešenje, ili pronaći rešenje u najmanjem mogućem broju poteza.



Slika 1. Igra Sokoban

Originalna postavka igre sastojala se iz 90 različitih nivoa[1], ali je kasnije od strane različitih autora osmišljeno više hiljada različitih Sokoban postavki. Programske implementacije Sokoban igre su tokom 80-tih godina 20. veka razvijene za većinu tadašnjih kućnih računara (poput računara Spectrum i C64), a kasnije i za PC računare. Danas je igra Sokoban dostupna i na internetu, u okviru velikog broja veb stranica posvećenih ovoj igri.

Pored „ručnog" rešavanja, danas postoji i značajno interesovanje istraživača za automatizacijom procesa rešavanja igre Sokoban [7, 8], pre svega zbog izuzetno velikog prostora pretrage, čak i u slučaju relativno jednostavnih nivoa. Igra Sokoban je razmatrana i u okviru teorijskog računarstva, gde je dokazano da je problem PSPACE-kompletan [9]. Ovo znači da se svaki problem koji je rešiv u polinomskom prostoru može polinomskom redukcijom svesti na neku Sokoban instancu. Ovo problem rešavanja igre Sokoban čini jednim od najtežih problema u PSPACE klasi.

Postojeći specijalizovani Sokoban rešavači su zasnovani na opštim i specifičnim metodama *informisane pretrage* (poput $A*$ [10], *IDA** [11], *FESS* [12], i sl.), unapređenim primenom specifičnih domenskih znanja. Ova domenska znanja se uglavnom sastoje u prepoznavanju tzv. *mrtvih stanja* (engl. *deadlocks*). Za stanje table za igru (koje je određeno trenutnim pozicijama igrača i kutija) kažemo da je mrtvo stanje, ako se iz tog stanja ne može stići do završnog stanja (tj. do stanja u kome su kutije postavljene na odredišne pozicije). Tipičan primer mrtvog stanja je slučaj kada je kutija postavljena uz zid od kog se više ne može odvojiti (jer igrač ne može da vuče kutije, već samo da ih gura), a u toj vrsti/koloni ne postoji odredišno polje na koje bi se kutija mogla postaviti. Pored ovog jednostavnog primera, postoji i mnogo drugih tipova mrtvih stanja koja nisu tako očigledna, ali su poznata iskusnim Sokoban igračima i razmatraju se u literaturi [13]. Automatsko prepoznavanje ovih mrtvih stanja može u značajnoj meri suziti prostor pretrage i time omogućiti efikasnije pronalaženje rešenja. Među najpoznatijim specijalizovanim Sokoban rešavačima izdvajamo *Jsoko*[2], *Sokolution*[3] i *Festival*[4], među kojima je ovaj poslednji, zasnovan na *FESS* algoritmu [12], trenutno najuspešniji i prvi koji je uspeo da reši svih 90 originalnih Sokoban instanci. Novije, teže instance Sokoban problema su i dalje van domašaja za automatske rešavače.

Pored rešavača koji se zasnivaju na implementaciji specifičnih metoda pretrage, postoje i rešavači koji problem rešavanja igre Sokoban svode na instance nekih opštih problema, poput *problema iskazne zadovoljivosti* (poznatog i kao *SAT problem* [14]) ili *problema zadovoljavanja ograničenja* (poznatog i kao *CSP problem* [15]), što omogućava primenu postojećih efikasnih alata za takve probleme. Među takvim rešavačima izdvaja se Sokoban rešavač[5] zasnovan na *copris*[6] jeziku za modelovanje. Ovaj jezik omogućava da se problem opiše na visokom nivou, a da se zatim isti automatski izrazi kao CSP problem i kao takav preda odgovarajućem rešavaču. Podrazumevana konfiguracija dobijeni CSP problem najpre svodi na SAT (koristeći alat *sugar* [16]), a zatim ga rešava pomoću SAT rešavača. Ovakvi pristupi su do sada takođe bili relativno uspešni.

Naš pristup prati drugi navedeni pravac istraživanja i razmatra mogućnost svođenja problema rešavanja igre Sokoban na problem automatskog planiranja, koji bi onda bio izražen kao *SMT problem* [17] i rešavan pomoću SMT rešavača. SMT problem predstavlja uopštenje SAT problema na logiku prvog reda, tj. razmatra se zadovoljivost formule prvog reda u odnosu na unapred zadatu teoriju. Za razliku od SAT pristupa, SMT pristup omogućava izražavanje problema na znatno višem nivou, čime se u velikoj meri zadržava originalna struktura problema koji se rešava. Takođe, SMT rešavači u sebi sadrže procedure odlučivanja koje su prilagođene tipovima ograničenja koji su podržani u datoj teoriji prvog reda, što omogućava efikasniji tretman ovih ograničenja. Za razliku od CSP rešavača, SMT rešavači imaju bolju podršku za opis i tretman složenijih iskaznih struktura datog problema, budući da logika prvog reda u sebi sadrži iskaznu logiku. Navedene činjenice nam daju razlog za optimizam kada je u pitanju upotrebljivost SMT rešavača u rešavanju ovakve vrste problema. Cilj ovog rada je preliminarno istraživanje u ovom pravcu, kao i razmatranje pravaca budućeg rada, na osnovu dobijenih preliminarnih rezultata.

Ostatak ovog rada ima sledeću strukturu. U poglavlju 2 dajemo kratak pregled osnovnih pojmova vezanih za SAT i SMT rešavače, kao i za problem automatskog planiranja na koji ćemo svoditi problem rešavanja igre Sokoban. Poglavlje 3 detaljno opisuje kodiranje problema koje je korišćeno u ovom istraživanju. Poglavlje 4 sadrži informacije o implementaciji i preliminarnim rezultatima. Najzad, u

[1] https://sokoban.info/

[2] https://www.sokoban-online.de/

[3] http://codeanalysis.fr/sokoban/

[4] https://festival-solver.site/

[5] https://cspsat.gitlab.io/copris-puzzles/sokoban/index.html

[6] http://bach.istc.kobe-u.ac.jp/copris/

poglavlju 5 dajemo zaključke i navodimo pravce budućeg rada.

2 AUTOMATSKO PLANIRANJE UZ POMOĆ SAT I SMT REŠAVAČA

2.1 SAT problem i SAT rešavači

U nastavku ovog teksta pretpostavljamo da su čitaocu poznati osnovni pojmovi iskazne logike [14]. Literal je ili iskazni atom (tj. iskazno slovo, poput $p, q, r, \ldots$), ili negacija iskaznog atoma (poput $\neg p, \neg q, \ldots$). *Klauza* je disjunkcija literala. Formula je u *konjunktivnoj normalnoj formi* (KNF) ako predstavlja konjunkciju klauza.

Valuacija je funkcija koja iskaznim atomima pridružuje istinotonosnu vrednost (*tačno* ili *netačno*). Svaka valuacija jedinstveno određuje *interpretaciju* proizvoljne formule F (takođe *tačno* ili *netačno*), imajući u vidu dobro poznatu semantiku iskaznih veznika. Za formulu kažemo da je *zadovoljiva*, ako postoji valuacija u kojoj je ta formula tačna. Problem ispitivanja zadovoljivosti iskazne formule poznat je kao *SAT problem* (engl. *SATisfiability*).

SAT problem je jedan od najpoznatijih NP-kompletnih problema [18] i svi poznati deterministički algoritmi za njegovo rešavanje su eksponencijalne vremenske složenosti u najgorem slučaju. Alati koji implementiraju ove algoritme nazivaju se SAT rešavači. Većina modernih SAT rešavača su zasnovani na *DPLL algoritmu* [19] iz 1962. godine koji se primenjuje na formule u KNF-u. U osnovi, DPLL algoritam je zasnovan na *pretrazi sa vraćanjem* (engl. *backtracking*), uz dodatak elemenata iskaznog rasuđivanja, poput *jediničnih klauza* i *čistih literala*. Za razliku od originalnog DPLL algoritma koji je bio implementiran rekurzivno, moderne implementacije su zasnovane na iterativnoj implementaciji, uz eksplicitnu reprezentaciju steka. Pored toga, prisutna su i brojna algoritamska poboljšanja (poput *analize konflikta*, *učenja klauza*, *nehronološkog vraćanja unazad* i *ponovnog započinjanja pretrage*), kao i implementaciona poboljšanja (poput *sheme dva posmatrana literala*). Ovako unapređeni algoritam se obično označava terminom *CDCL* (engl. *conflict driven clause learning* [20]). Moderni CDCL SAT rešavači su u stanju da rešavaju probleme koji uključuju na desetine hiljada iskaznih atoma, kao i na stotine hiljada klauza. Zahvaljujući tako impresivnim performansama, mnogi problemi se danas mogu rešavati svođenjem na SAT problem, umesto da se razmatraju domenski specifična rešenja. Ovakav pristup dao je značajne rezultate kako u verifikaciji hardvera i softvera, tako i u problemima automatskog planiranja, kombinatornog dizajna, problemima raspoređivanja, diskretnoj geometriji i sl.

Neki od najpopularnijih SAT rešavača su *minisat*[7], *glucose*[8], *lingeling*[9], itd.

[7] http://minisat.se/
[8] https://www.labri.fr/perso/lsimon/glucose/
[9] http://fmv.jku.at/lingeling/

2.2 SMT problem i SMT rešavači

Zadovoljivost formula se može razmatrati i u logici prvog reda, gde se umesto iskaznih atoma pojavljuju *atomi prvog reda* [5]. Atomi prvog reda dobijaju se primenom predikatskih simbola na termove. Termovi se, sa druge strane, grade od promenljivih i konstanti, na koje se primenjuju funkcijski simboli. Dodatno, promenljive se mogu kvantifikovati (univerzalnim ili egzistencijalnim kvantifikatorom). Ukoliko formula ne sadrži promenljive, tada je nazivamo *baznom formulom*.

Predikatski simboli se mogu interpretirati relacijama odgovarajuće arnosti nad unapred fiksiranim domenom, dok se funkcijski simboli mogu interpretirati funkcijama (operacijama) odgovarajuće arnosti nad tim domenom. U slučaju baznih formula, interpretacija formule biće u potpunosti određena interpretacijom funkcijskih i predikatskih simbola koji se u njoj pojavljuju. Formula će biti *zadovoljiva*, ako postoji interpretacija simbola koji se pojavljuju u formuli takva da je formula u njoj tačna. Uobičajeno je da pretpostavljamo da interpretacija simbola koji se pojavljuju u formuli nije potpuno proizvoljna, već da je na neki način fiksirana (najčešće zadavanjem skupa aksioma koje moraju biti zadovoljene). Tada govorimo o *teoriji prvog reda*, a interpretacije koje zadovoljavaju dati skup aksioma nazivamo *modelima* te teorije. Za formulu kažemo da je *zadovoljiva u odnosu na teoriju* ako postoji model te teorije u kome je data formula tačna. Problem ispitivanja zadovoljivosti u odnosu na teoriju naziva se *SMT problem* (engl. *Satisfiability Modulo Theory* [17]). Alati koji implementiraju procedure za rešavanje SMT problema nazivaju se *SMT rešavači*.

Većina modernih SMT rešavača su zasnovani na takozvanom lenjom pristupu [17] – atomi prvog reda se najpre apstrahuju iskaznim atomima i pomoću SAT rešavača se proverava iskazna zadovoljivost tako dobijene formule. Za dobijeni iskazni model (ako postoji) ispituje se da li je odgovarajuća konjunkcija literala prvog reda zadovoljiva u datoj teoriji, korišćenjem posebne procedure odlučivanja. Drugim rečima, SMT rešavač se sastoji iz SAT rešavača (najčešće CDCL zasnovanog) i specifične procedure odlučivanja za teoriju od interesa, koju nazivamo i *teorijski rešavač*. Na ovaj način, SMT rešavači kombinuju efikasnost SAT pretrage sa specifičnim rezonovanjem u teoriji koja se razmatra.

Prednost SMT rešavača u odnosu na SAT rešavača je u njihovoj izražajnosti. Mnoge uslove i ograničenja je lakše izraziti, na primer, pomoću aritmetičkih relacija nego na jeziku iskazne logike. Teorije koje se razmatraju su obično izabrane tako da nalaze svoju primenu u verifikaciji softvera, što je glavni domen primene SMT rešavača. Ipak, u poslednje vreme, SMT rešavači se sve više koriste i u drugim oblastima (poput problema zadovoljavanja ograničenja, automatskog planiranja, i sl.).

Neki od najpopularnijih SMT rešavača su *Z3*[10], *CVC4*[11],

[10] https://github.com/Z3Prover/z3
[11] https://cvc4.github.io/

MathSAT5[12], itd.

2.3 Automatsko planiranje

Automatsko planiranje [21] podrazumeva razvoj procedura za automatsko pronalaženje plana, tj. niza akcija koje je potrebno primeniti da bi se postigao zadati cilj. Formalno, problem planiranja je uređena četvorka $P = (V, S_0, O, G)$, gde je:

- V – skup promenljivih koje uzimaju vrednosti iz odgovarajućih domena
- S_0 – početno stanje (pod *stanjem* podrazumevamo mapiranje koje promenljivama iz V pridružuje vrednosti iz njihovih domena)
- O – skup operatora oblika $o = (C, E)$, gde je C *preduslov* koji mora da zadovolji neko stanje S (što označavamo sa $C(S)$) da bi na njega mogao biti primenjen operator o, a E je skup *efekata*, pri čemu je svaki efekt oblika:

  $$F \Rightarrow \{x_1 := v_1, x_2 := v_2, \ldots, x_n := v_n\}$$

  gde je F (opcioni) uslov za izvršenje efekta koji se sastoji iz niza dodela $x_i := v_i$ ($v_i \in V$). Operator o se primenjuje na neko stanje S (što označavamo sa $o(S)$) tako što se svaki efekat ovog operatora čiji je preduslov F ispunjen u stanju S izvršava, primenom odgovarajućih dodela, čime se dobija novo stanje S' (pretpostavlja se da ove dodele nisu međusobno u koliziji, u suprotnom operator nije moguće primeniti).
- G je uslov cilja, tj. uslov koji je potrebno da ispunjava završno stanje dobijeno nakon primene plana.

Plan koji je rešenje problema planiranja P je bilo koji niz operatora $o_1, o_2, \ldots, o_N$ ($o_i = (C_i, E_i) \in O$), takav da važi:

- $C_i(S_{i-1})$ za $1 \leq i \leq N$
- $S_i = o_i(S_{i-1})$, za $1 \leq i \leq N$
- $G(S_N)$

Pod dužinom plana podrazumevamo broj operatora N iz kojih se plan sastoji. Problem ispitivanja da li postoji plan (proizvoljne dužine) za dati problem planiranja je PSPACE-kompletan [22]. Ukoliko fiksiramo dužinu plana N, tada je problem ispitivanja da li postoji plan dužine N NP-kompletan i može se svoditi na SAT i SMT probleme.

3 SOKOBAN KAO PROBLEM PLANIRANJA

Igra sokoban se sasvim prirodno može posmatrati kao problem planiranja, gde je stanje određeno pozicijom igrača i kutija na tabli za igru, a operatori odgovaraju mogućim potezima igrača (*gore*, *dole*, *levo*, *desno*), pri čemu za svaki operator postoje odgovarajući preduslovi koji moraju biti ispunjeni da bi taj operator mogao da se primeni. Na primer, ne možemo pomeriti igrača na gore ako je iznad igrača zid, ili je iznad igrača kutija iznad koje je zid ili druga kutija. Slični uslovi važe i za ostale poteze. Uslov cilja se sastoji u tome da se pozicije kutija poklapaju sa unapred zadatim odredišnim pozicijama na tabli.

Formalno, pretpostavimo da imamo tablu dimenzija $m \times n$. Svaka pozicija na tabli određena je svojim koordinatama (x, y), gde je $x \in \{0, 1, \ldots, n-1\}$ indeks kolone, dok je $y \in \{0, 1, \ldots, m-1\}$ indeks vrste. Najpre je potrebno opisati okruženje za igru, tj. pozicije na tabli na kojima se nalazi zid, kao i pozicije koje predstavljaju odredišna polja (ova postavka je fiksna i ne menja se tokom igre, pa samim tim ne predstavlja deo stanja). S obzirom da je tabla za igru dvodimenziona, raspored zidova i odredišnih polja je najlakše zadati matricom logičkih vrednosti[13], gde će vrednost nekog polja matrice biti *true* ako i samo ako je na tom polju zid, odnosno odredište. Međutim, kako SMT rešavači nemaju direktnu podršku za rad sa matricama, umesto matrica koristićemo jednodimenzione nizove logičkih vrednosti dužine $m \cdot n$, pri čemu će polju matrice na poziciji (x, y) odgovarati element niza sa indeksom $n \cdot y + x$. Odgovarajuće nizove nazovimo redom $wall$ i $home$.

Stanje igre nakon primene k-tog operatora biće opisano celobrojnim promenljivama X_k i Y_k koje predstavljaju koordinate igrača, kao i matricom logičkih vrednosti koja određuje pozicije kutija. Slično kao i malopre, matricu logičkih vrednosti predstavljaćemo jednodimenzionim nizom logičkih vrednosti dužine $m \cdot n$. Ovu promenljivu označimo sa box_k.

Kodiranje okruženja. Okruženje, tj. konfiguracija table za igru se kodira zadavanjem vrednosti promenljivih $wall$ i $home$. S obzirom da su u pitanju nizovske promenljive, njihova vrednost se zadaje eksplicitnim zadavanjem vrednosti svih elemenata niza.

Kodiranje početnog stanja. Početno stanje S_0 se kodira zadavanjem vrednosti promenljivih X_0 i Y_0, kao i promenljive box_0, na osnovu početne situacije na tabli za igru.

Kodiranje plana. Plan unapred fiksirane dužine N biće opisan celobrojnim promenljivama $move_1, move_2, \ldots, move_N$, pri čemu $move_k \in \{0, 1, 2, 3\}$ predstavlja operator (tj. potez) koji je primenjen u k-tom koraku (0-gore, 1-dole, 2-levo i 3-desno). Za svaki od mogućih operatora $op \in \{0, 1, 2, 3\}$, neka je formulom $C_{op}[k]$ izražen preduslov za izvršenje tog operatora u k-tom koraku, i neka su formulom $E_{op}[k]$ izraženi efekti izvršenja tog operatora u k-tom koraku. Semantika operatora op u k-tom koraku izražena je sledećim formulama:

- $(move_k = op) \Rightarrow C_{op}[k]$ – ako je u k-tom koraku izabran operator op, tada mora da važi preduslov ovog operatora $C_{op}[k]$
- $(move_k = op) \Rightarrow E_{op}[k]$ – ako je u k-tom koraku izabran operator op, tada moraju biti izvršeni efekti

[12] `https://mathsat.fbk.eu/`

[13] Elementi logičke matrice imaju vrednost $true$ ili $false$.

ovog operatora $E_{op}[k]$.

Kodiranje plana se sada sastoji iz konjunkcije gornjih formula za svako $op \in \{0, 1, 2, 3\}$ i svako $k \in \{1, 2, \ldots, N\}$. Ilustracije radi, prikažimo kodiranje formula $C_{op}[k]$ i $E_{op}[k]$ za $op = 0$ (tj. potez igrača „na gore"). Formula $C_0[k]$ biće sledećeg oblika:

$$\left(\begin{array}{c} Y_{k-1} > 0 \wedge \\ \neg wall[(Y_{k-1}-1)\cdot n + X_{k-1}] \wedge \\ box_{k-1}[(Y_{k-1}-1)\cdot n + X_{k-1}] \Rightarrow \\ Y_{k-1} > 1 \wedge \\ \neg box_{k-1}[(Y_{k-1}-2)\cdot n + X_{k-1}] \wedge \\ \neg wall[(Y_{k-1}-2)\cdot n + X_{k-1}] \end{array}\right)$$

Intuitivno, da bismo mogli da odigramo potez „na gore", igrač ne sme biti u nultoj vrsti, iznad igrača ne sme biti zid, a ako je iznad igrača kutija, tada iznad nje ne sme biti ni zid ni kutija. Formula $E_0[k]$ biće sledećeg oblika:

$$\begin{array}{c} Y_k = Y_{k-1} - 1 \wedge \\ X_k = X_{k-1} \wedge \\ \left(\begin{array}{c} \neg box_{k-1}[(Y_{k-1}-1)\cdot n + X_{k-1}] \Rightarrow \\ box_k = box_{k-1} \end{array}\right) \wedge \\ \left(\begin{array}{c} box_{k-1}[(Y_{k-1}-1)\cdot n + X_{k-1}] \Rightarrow \\ box_k = box_{k-1}|_{(Y_{k-1}-1)\cdot n + X_{k-1}\uparrow} \end{array}\right) \end{array}$$

pri čemu oznaka $box_{k-1}|_{(Y_{k-1}-1)\cdot n + X_{k-1}\uparrow}$ označava isti niz logičkih vrednosti kao i box_{k-1}, s tom razlikom što je vrednost na indeksu $(Y_{k-1}-1)\cdot n + X_{k-1}$ jednaka $false$, a vrednost na indeksu $(Y_{k-1}-2)\cdot n + X_{k-1}$ je jednaka $true$.[14] Intuitivno, efekat poteza „na gore" je u tome da se y koordinata igrača smanjuje za jedan, dok x koordinata ostaje nepromenjena. Pritom, ako je iznad igrača bila kutija, ona se pomera za jedno mesto na gore, dok u suprotnom raspored kutija na tabli ostaje nepromenjen.

Kodiranje cilja. Cilj G je predstavljen sledećom formulom:

$$box_N = home$$

Dakle, pozicije kutija u završnom stanju moraju se poklapati sa pozicijama odredišnih polja.

Pronalaženje najkraćeg plana. Neka je ϕ_N gore opisana formula za fiksiranu dužinu plana N. Pronalaženje plana najmanje dužine se svodi na sukcesivno ispitivanje zadovoljivosti formula $\phi_1, \phi_2, \ldots$ do pronalaska prve zadovoljive formule u ovom nizu.

4 IMPLEMENTACIJA I EVALUACIJA

Prethodno kodiranje visokog nivoa je dalje potrebno zapisati na jeziku rešavača koji će biti korišćen. U slučaju SMT rešavača, dodatno se postavlja pitanje izbora teorije prvog reda koja je najpogodnija za kodiranje navedenih uslova. Prvi izbor je svakako *teorija celobrojne linearne aritmetike* [17], čiji jezik podržava linearna ograničenja nad celobrojnim domenom. Na jeziku ove teorije se mogu kodirati uslovi vezani za domene celobrojnih promenljivih X_k, Y_k i $move_k$, kao i linearni izrazi oblika $n \cdot y + x$ koji se koriste za određivanje indeksa u nizovima. Za same nizove, sa druge strane, može se koristiti *teorija nizova* [17] koja se u verifikacijskim primenama tipično koristi za apstrakciju memorije. Jezik ove teorije podržava funkcijske simbole $select$ i $store$ koji se, redom, koriste za čitanje i upis vrednosti elementa niza na datom indeksu. Pritom, izraz $select(a, i)$ predstavlja vrednost i-tog elementa niza a, dok izraz $store(a, i, v)$ predstavlja novi niz koji se dobija tako što se u nizu a na poziciju i upiše vrednost v, dok se ostali elementi ne menjaju. Na ovaj način se, recimo, uslov:

$$box_k = box_{k-1}|_{(Y_{k-1}-1)\cdot n + X_{k-1}\uparrow}$$

može jednostavno izraziti na sledeći način:

$$box_k = store(store(box_{k-1}, (Y_{k-1}-1)\cdot n + X_{k-1}, 0), (Y_{k-1}-2)\cdot n + X_{k-1}, 1)$$

Dobra strana SMT rešavača je da oni podržavaju i tzv. *kombinaciju teorija* [17], tj. moguće je u istoj formuli imati mešovita ograničenja koja istovremeno potiču iz većeg broja podržanih teorija. Primera radi, gornji uslov je izražen nad kombinacijom teorije aritmetike i teorije nizova. Na žalost, ovakvo kodiranje, iako veoma elegantno, nije dalo zadovoljavajuće rezultate. Rešavac Z3 je na ovaj način uspeo da reši isključivo neke veoma lake instance čije rešavanje nije zahtevalo više od 20 poteza.[15]

Drugi izbor je *teorija bitvektora* [17]. Ova SMT teorija se u verifikacijskim primenama koristi za modelovanje celobrojnih tipova sa fiksiranim brojem bitova u svom zapisu koji postoje u većini tipičnih programskih jezika (poput *int* tipa u C-u). U ovoj teoriji, izrazi se interpretiraju nizovima bitova konačnih dužina. Podržane su aritmetičke operacije (sa prekoračenjem), logičke i relacione, kao i bitovske operacije (poput pomeranja u levo i desno). Pored aritmetike, teorija bitvektora omogućava i reprezentaciju nizova logičkih vrednosti koji se mogu predstaviti prosto kao bitvektori odgovarajućih dužina. Otuda se teorija bitvektora može iskoristiti za kodiranje našeg problema, iako kodiranje nije tako elegantno i jednostavno kao u prethodnom slučaju. Na primer, izdvajanje elementa niza bi se ovde svodilo na očitavanje bita na datoj bitskoj poziciji u odgovarajućem bitvektoru, što se svodi na upotrebu bitovskih operatora (kao u jeziku C). Ipak, ovim kodiranjem postignuti su znatno bolji rezultati. Rešavač Z3 je uspeo da reši, pored lakih, i izvestan broj srednje teških instanci, za čije je rešavanje bilo potrebno do 100 poteza. Teže instance su i dalje ostale van domašaja.[16]

Razlog za bolje ponašanje teorije bitvektora, po mišljenju autora, leži u konačnosti domena koji se u ovoj teoriji razmatraju. Naime, svaki bitvektor fiksirane dužine može uzeti

[14] Konkretno kodiranje ovog uslova zavisiće od izbora SMT teorije koja se koristi.

[15] Pored rešavača Z3, razmatran je i rešavač CVC4, ali je njegovo ponašanje bilo još lošije, te smo ga isključili iz dalje evaluacije.

[16] Detalji implementacije i evaluacije mogu se naći na lokaciji: `https://github.com/milanbankovic/smt_sokoban`.

konačno mnogo različitih vrednosti, što nije slučaj sa celobrojnim tipom u teoriji aritmetike, čiji model je *beskonačni* skup celih brojeva $\mathbb{Z}$. Konačnost domena čini teorijski rešavac za teoriju bitvektora znatno efikasnijim i bolje prilagođenim ovakvoj vrsti problema. Na primer, bitvektori kojima se predstavljaju promenljive $move_k$ mogu biti dužine 2, s obzirom da imamo samo 4 moguća poteza. Ovakvi bitvektori se znatno jednostavnije mogu razmatrati u okviru SMT rešavača od celobrojnih promenljivih koje mogu biti interpretirane bilo kojim elementom skupa $\mathbb{Z}$.

5 ZAKLJUČAK I DALJI RAD

U ovom radu razmatrana je primena SMT rešavača u automatskom pronalaženju rešenja igre Sokoban, svođenjem na problem automatskog planiranja. Razmatrana su dva kodiranja, jedno zasnovano na kombinaciji teorije celobrojne aritmetike i teorije nizova i drugo, zasnovano na teoriji bitvektora. Pritom, ovo drugo kodranje dalo je ohrabrujuće rezultate tokom preliminarne evaluacije na lakim i srednje teškim instancama. Ipak, kako bi se rezultati zasnovani na ovom pristupu približili modernim specijalizovanim Sokoban rešavačima, neophodna su dodatna poboljšanja kodiranja. Ova poboljšanja mogu ići u dva pravca. Prvi podrazumeva kodiranje uslova vezanih za izbegavanje mrtvih stanja, čime se značajno može smanjiti prostor pretrage. Drugi pristup podrazumeva ukrupnjavanje poteza. Naime, često je neophodno da igrač pređe dug put do kutije koju treba pomeriti. Suštinski, samo pomeranje kutije je potez koji zaista menja stanje, jer pomeranje igrača samo za sebe nema efekta. Ipak, u našem kodiranju svako pomeranje „u prazno" se takođe broji kao potez, što planove koje se razmatraju može učiniti veoma dugim, naročito u slučajevima kada je tabla za igru velikih dimenzija. Umesto toga, čitav put do kutije i njeno pomeranje se može smatrati jednim potezom, čime bi planovi postali znatno kraći. Sa druge strane, kodiranje plana bi postalo složenije. Detaljnijom evaluacijom, koja ostaje predmet budućeg rada, treba utvrditi u kojoj meri bi ovakav pristup proces rešavanja učinio efikasnijim. Drugi zadatak budućeg rada je i detaljno poređenje sa drugim aktuelnim pristupima.

Literatura

[1] Diogo M Costa. Computational complexity of games and puzzles. *arXiv preprint arXiv:1807.04724*, 2018.

[2] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[3] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[4] Mariusz Flasiński. Symbolic artificial intelligence. In *Introduction to Artificial Intelligence*, pages 15–22. Springer, 2016.

[5] Alan JA Robinson and Andrei Voronkov. *Handbook of automated reasoning*, volume 1. Elsevier, 2001.

[6] Dorit Dor and Uri Zwick. Sokoban and other motion planning problems. *Computational Geometry*, 13(4):215–228, 1999.

[7] Andreas Junghanns and Jonathan Schaeffer. Sokoban: A challenging single-agent search problem. In *In IJCAI Workshop on Using Games as an Experimental Testbed for AI Reasearch*. Citeseer, 1997.

[8] Adi Botea, Martin Müller, and Jonathan Schaeffer. Using abstraction for planning in sokoban. In *International Conference on Computers and Games*, pages 360–375. Springer, 2002.

[9] Joseph Culberson. Sokoban is pspace-complete. 1997.

[10] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[11] Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

[12] Yaron Shoham and Jonathan Schaeffer. The fess algorithm: A feature based approach to single-agent search. In *2020 IEEE Conference on Games (CoG)*, pages 96–103. IEEE, 2020.

[13] Jean Persi Boelter. Learning deadlocks in sokoban. 2018.

[14] 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.

[15] Krzysztof Apt. *Principles of Constraint Programming*. Cambridge University Press, New York, NY, USA, 2003.

[16] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. Compiling finite linear CSP into SAT. *Constraints*, 14(2):254–272, 2009.

[17] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, chapter 26, pages 825–885. IOS Press, 2009.

[18] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[19] Martin Davis, George Logemann, and Donald Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[20] Joao Marques-Silva, Ines Lynce, and Sharad Malik. Conflict-Driven Clause Learning SAT Solvers. In *Handbook of Satisfiability*, chapter 4, pages 131–155. IOS Press, 2009.

[21] Jussi Rintanen. Planning and sat. *Handbook of Satisfiability*, 185:483–504, 2009.

[22] Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.