Implementacija Sudoku rešavača korišćenjem Algoritma igrajućih pokazivača

Seminarski rad u okviru kursa
Konstrukcija i Analiza algoritama 2
Matematički fakultet

Petar Đorđević

13. avgust 2024.

Sažetak

Ovaj rad istražuje Algoritam igrajućih pokazivača (*Dancing Links Algorithm - DLX*), koji je efikasna metoda za rešavanje problema pokrivanja tačaka, specifično za rešavanje problema tačnog omotača. Algoritam, koji je osmislio Donald Knut [4], omogućava brzu i efikasnu manipulaciju matrica pokrivanja. Ova tehnika je naročito pogodna za rešavanje složenih kombinatornih problema kao što je raspoređivanje poslova, logičke rešetke, puzzle i slično. U okviru ovog rada, implementiraće se Sudoku rešavač kao konkretan primer primene Algoritma igrajućih pokazivača (DLX).

Sadržaj

1 Uvod **2**

2 Problem tačnog omotača **2**
2.1 Matrica pokrivanja 3

3 Algoritam igrajućih pokazivača **3**
3.1 Operacije nad dvostruko ulančanim listama 3
3.2 Struktura igrajućih linkova 4
3.3 Operacije nad igrajućim pokazivačima 5

4 Sudoku **6**
4.1 Svođenje na problem tačnog pokrivača 7

5 Zaključak **8**

Literatura **8**

1 Uvod

Problem tačnog pokrivača (*Exact Cover Problem*) je klasičan kombinatorni problem odabira podskupova iz date kolekcije tako da svaki element univerzalnog skupa bude tačno jednom pokriven [1]. Rešavanje ovog problema ima primenu u raznim oblastima kao što su teorija grafova, optimizacija, veštačka inteligencija i računarstvo. Problem tačnog pokrivača je NP-težak, što znači da je izuzetno složen za rešavanje, te zahteva najbolji mogući algoritam za efikasno pronalaženje rešenja.

Jedan od najpoznatijih algoritama za rešavanje ovih problema je Algoritam igrajućih pokazivača (*Dancing Links Algorithm - DLX*), koji je osmislio Donald Knuth. DLX koristi strukturu podataka poznatu kao igrajući pokazivači koja optimizuje operacije dodavanja, uklanjanja i pretraživanja elemenata u matricama pokrivanja.

Sudoku je popularna logička slagalica koja zahteva popunjavanje 9x9 mreže brojevima od 1 do 9, uz poštovanje pravila da se svaki broj mora pojaviti tačno jednom u svakom redu, koloni i 3x3 podmreži. Problem rešavanja Sudokua može se preformulisati kao problem tačnih pokrivača, što ga čini idealnim za primenu DLX-a. Ovo preformulisanje omogućava da se koristi DLX za efikasno pronalaženje rešenja, čak i za najteže zagonetke.

Cilj ovog rada je da implementira Sudoku rešavač koristeći Algoritam igrajućih pokazivača i da demonstrira efikasnost ovog algoritma. Pored toga, rad će evaluirati performanse DLX-a na različitim primerima Sudoku zagonetki.

2 Problem tačnog omotača

Problem tačnog pokrivača (*Exact Cover Problem - ECP*) predstavlja ključnu podvrstu problema zadovoljivosti ograničenja (*Constraint Satisfaction Problems - CSP*), gde je cilj pronalaženje podskupa elemenata koji tačno pokrivaju dati skup. Svaki podskup se može posmatrati kao klauza, a tačno pokrivanje zahteva da svaka klauza bude zadovoljena tačno jednom.

ECP se može definisati na sledeći način: Dat je univerzalni skup U i kolekcija S podskupova U. Cilj je pronaći podskup $S' \subseteq S$ takav da su svi elementi iz U tačno jednom pokriveni, odnosno svaki element iz U pripada tačno jednom podskupu iz S'.

$$\mathcal{S}' \subseteq \mathcal{S} \quad \wedge \quad \left(\forall S_1, S_2 \in \mathcal{S}',\ S_1 \neq S_2 \implies S_1 \cap S_2 = \emptyset\right) \quad \wedge \quad \bigsqcup_{S' \in \mathcal{S}'} S' = U$$

ECP se može svesti na CSP [2] na sledeći način: S predstavlja izbore koje možemo napraviti a U ograničenja nad tim izborima. Pošto je ova redukcija moguća znamo da je to NP-težak problem.

Postoje različiti pristupi rešavanju ECP-a, među kojima su i algoritmi koji koriste tehniku pretraživanja unazad, kao što su algoritam tačne pretrage i algoritam podeli pa vladaj. Pored toga, neki od najefikasnijih algoritama za rešavanje problema tačnog pokrivača su bazirani na pretraživanju uz upotrebu heuristika, kao što su gramzivi algoritmi i algoritmi zasnovani na tačnom pokrivanju kontraprimera (*backtracking*). Ovi algoritmi kombinuju preciznost i efikasnost kako bi pronašli optimalna rešenja ili dobra približna rešenja problema tačnog pokrivača.

2.1 Matrica pokrivanja

ECP se može predstaviti matricom incidencije: svaki red u matrici predstavlja jedan element iz skupa S dok kolona predstavlja element iz skupa U. U takvoj matrici, M_{ij} je 1 ukoliko se element U_j nalazi u podskupu S_i, a 0 obratno. ECP se rešava tako što se "uklone" svi redovi tako da u svakoj koloni postoji tačno jedno polje koje ima vrednost 1. U tabeli 1 možete videti primer matrice incidencije za sledeći problem:

$$U = \{A, B, C, D, E, F, G\}$$
$$S = \{\{C, E, F\}, \{A, D, G\}, \{B, C, F\}, \{A, D\}, \{B, G\}, \{D, E, G\}\}$$
$$S' = \{\{C, E, F\}, \{A, D\}, \{B, G\}\}$$

Tabela 1: Primer matrice incidencije

A	B	C	D	E	F	G
0	0	1	0	1	1	0
1	0	0	1	0	0	1
0	1	1	0	0	1	0
1	0	0	1	0	0	0
0	1	0	0	0	0	1
0	0	0	1	1	0	1

Ovakva matrica se zove matrica pokrivanja. Pretraga matrice pokrivanja za ECP često može biti spora zbog činjenice da se moraju pretraživati sva polja matrice kako bi se pronašla ona koja sadrže 1. Kako se matrica pokrivanja često sastoji od većeg broja redova i kolona, mnogo vremena se može potrošiti na pretragu polja koja sadrže 0 pre nego što se dođe do onih koja sadrže 1. Ovo može usporiti izvršavanje algoritma za rešavanje ECP-a, posebno u slučaju kada matrica pokrivanja ima velike dimenzije.

3 Algoritam igrajućih pokazivača

Algoritam igrajućih pokazivača (*Dancing Links Algorithm - DLX*), poznatiji kao "Algoritam X", je efikasan algoritam, definisan od strane Donalda Knutha u radu objavljenom 2000. godine [4], koji se koristi za rešavanje ECS-a i sličnih kombinatornih problema. Ovaj algoritam koristi specifičnu strukturu podataka nazvanu igrajući pokazivači koja korišćenjem dvostruko povezanih listi optimizuje operacija dodavanja, uklanjanja i pretraživanja elemenata u matricama pokrivanja. Zasnovana je na ideji da se optimizuje pretraga matrice pokrivanja tako što se uvede matrica dvostruko uvezanih listi i time eliminiše obilazak polja u matrici koja imaju vrednost 0.

3.1 Operacije nad dvostruko ulančanim listama

Neka je x pokazivač na element dvostruko ulančane liste. Takođe, neka $L[x]$ i $R[x]$ predstavljaju pokazivač na element levo, odnosno desno od x. Možemo definisati dve operacije:

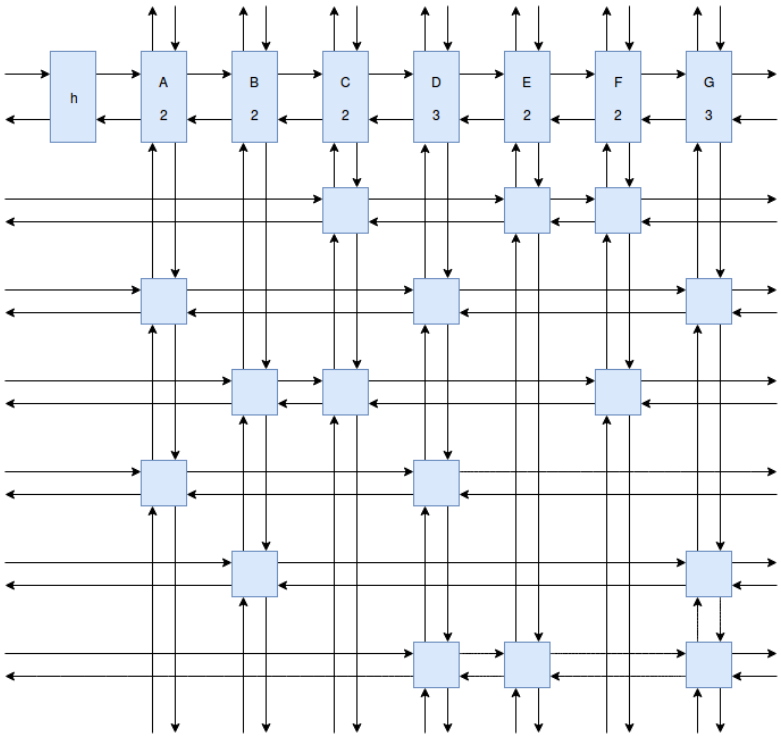
$$\texttt{hide(x)} := \begin{cases} L[R[x]] \leftarrow L[x] \\ R[L[x]] \leftarrow R[x] \end{cases} \qquad \texttt{unhide(x)} := \begin{cases} L[R[x]] \leftarrow x \\ R[L[x]] \leftarrow x \end{cases}$$

Operacija $hide(x)$ uklanja element iz liste, dok ga $unhide(x)$ vraća u listu ukoliko je prethodno izbačen, a u suprotnom ne radi ništa. Uzastopno izvršavanje $hide(x)$ pa $unhide(x)$ vraća listu u prvobitno stanje, i obe operacije su idempotentne (kod uzastopnih poziva iste operacije nad istim elementom, samo prvi može promeniti listu, naredni pozivi ne rade ništa).

Usled gore navedenih osobina, dvostruko ulančane liste su pogodne za pamćenje stanja i simulaciju backtrackinga.

3.2 Struktura igrajućih linkova

Osnovna ideja ovog algoritma je korišćenje cikličnih dvostruko ulančanih listi kako bi se predstavila matrica pokrivanja. Matrica se formira tako što se svaki red predstavlja jednim elementom koji treba pokriti, a svaka kolona predstavlja jedan mogući način da se taj element pokrije. Cilj je da se odabere kombinacija kolona (elementi) koja pokriva sve redove, pri čemu svaki element treba biti pokriven tačno jednom. Svaki element u matrici je jedan čvor ciklične liste, a lista se formira ne samo po redovima već i po kolonama tako da jedan element pripada zapravo dvema cikličnim dvostruko ulančanim listama: lista kolone i lista vrste. Dodatno, matrica obično sadrži i red zaglavlja i cvor koren. Red zaglavlja se koristi za identifikaciju kolona u matrici, dok se cvor koren koristi kao ulazna tačka u strukturu podataka, omogućavajući jednostavan pristup početnim tačkama pretrage. Na slici 1 ilustrovano je inicijalno stanje igrajućih linkova primera iz tabele 1.



Slika 1: Ilustacija inicijalnog stanja igrajućih linkova primera iz tabele 1

Jedan čvor u ovim listama se obično sastoji od pet polja:

- $L[x]$: Pokazivač na levi susedni čvor.
- $R[x]$: Pokazivač na desni susedni čvor.
- $U[x]$: Pokazivač na gornji susedni čvor.
- $D[x]$: Pokazivač na donji susedni čvor.
- $C[x]$: Pokazivač na "kolonu" čvora, što predstavlja zaglavlje kolone u matrici.

Pored ovih polja, čvor unutar zaglavlja takođe ima i polje $size[x]$ koje sadrži broj elemenata u koloni, i polje $name[x]$ koje sadrži labelu (ime) kolone. Ova polja nisu neophoda, ali pojednostavljuju implementaciju strukture.

Algoritam DLX koristi backtracking strategiju, gde se prvo odabire jedan od mogućih elemenata (kolona) za pokrivanje i proverava se da li je ova kombinacija valjana. Ako jeste, algoritam nastavlja dalje pretragu u istom pravcu, inače se vraća korak unazad (backtrack) i bira drugi element za pokrivanje. Ovaj proces se ponavlja sve dok se ne pronađe rešenje ili se iscrpe sve mogućnosti. Algoritam se izvodi rekurzivno pretragom u dubinu kako bi se obišli svi mogući putevi u pretrazi.

3.3 Operacije nad igrajućim pokazivačima

Oslanjajući se na osobine funkcija $hide(x)$ i $unhide(x)$ koje su malo pre definisane, možemo slične funkcije definisati nad čvorovima tako da se mogu izvršavati i horizontalno i vertikalno.

Horizontalni slučaj:

$$\texttt{hideh(x)} := \begin{cases} L[R[x]] \leftarrow L[x] \\ R[L[x]] \leftarrow R[x] \end{cases} \qquad \texttt{unhideh(x)} := \begin{cases} L[R[x]] \leftarrow x \\ R[L[x]] \leftarrow x \end{cases}$$

Vertikalni slučaj, proširen činjenicom da zaglavlje prati broj aktivnih elemenata u koloni:

$$\texttt{hidev(x)} := \begin{cases} U[D[x]] \leftarrow U[x] \\ D[U[x]] \leftarrow D[x] \\ C[x] \leftarrow C[x] - 1 \end{cases} \qquad \texttt{unhidev(x)} := \begin{cases} U[D[x]] \leftarrow x \\ D[U[x]] \leftarrow x \\ C[x] \leftarrow C[x] + 1 \end{cases}$$

Nakon što smo definisali osnovne operacije, možemo ih koristiti za implementaciju funkcija `cover` i `uncover`. Funkcija `cover(x)` sakriva kolonu x i sve redove koji je presecaju. Funkcija `uncover(x)` obnavlja kolonu x i sve redove koji je presecaju.

$$\texttt{cover(x)} = \begin{cases} \texttt{hideh(x)} \\ \text{For } (\texttt{p} \leftarrow \texttt{D[x]}; \quad \texttt{p} \neq \texttt{x}; \quad \texttt{p} \leftarrow \texttt{D[p]}) \\ \quad \text{For } (\texttt{r} \leftarrow \texttt{R[p]}; \quad \texttt{r} \neq \texttt{p}; \quad \texttt{r} \leftarrow \texttt{R[r]}) \\ \quad\quad \texttt{hidev(r)} \end{cases}$$

Funkcija `cover(x)` prvo sakriva kolonu x pomoću `hideh(x)`. Zatim prolazi kroz sve redove ispod x koristeći pokazivač `p`. Unutar svakog reda, prolazi kroz sve desne elemente koristeći pokazivač `r` i sakriva ih pomoću `hidev(r)`.

```
uncover(x) = { For (p ← U[x];  p ≠ x;  p ← U[p])
               {   For (l ← L[p];  l ≠ p;  l ← L[l])
               {     unhidev(l)
               { unhideh(x)
```

Funkcija `uncover(x)` prvo prolazi kroz sve redove iznad x koristeći pokazivač `p`. Unutar svakog reda, prolazi kroz sve leve elemente koristeći pokazivač `l` i obnavlja ih pomoću `unhidev(l)`. Na kraju, obnavlja kolonu x pomoću `unhideh(x)`.

Sada kada smo definisali osnovne operacije i funkcije `cover` i `uncover`, možemo preći na implementaciju samog DLX algoritma. Algoritam koristi rekurzivnu funkciju `search` koja pokušava da pronađe rešenje problema tačnog pokrivanja.

```
search(k) = { if R[head] = head
            {   print solution and return
            { else
            {   c ← choose column
            {   cover(c)
            {   For (r ← D[c];  r ≠ c;  r ← D[r])
            {     set solution[k] = r
            {     For (j ← R[r];  j ≠ r;  j ← R[j])
            {       cover(j)
            {     search(k+1)
            {     For (j ← L[r];  j ≠ r;  j ← L[j])
            {       uncover(j)
            {   uncover(c)
```

Ova funkcija prvo proverava da li je problem rešen (ako je zaglavlje kolone jedino preostalo). Ako jeste, ispisuje rešenje i vraća se. Ako nije, bira se kolona c koja će biti pokrivena. Nakon što se kolona c pokrije, algoritam prolazi kroz sve redove r u toj koloni i pokušava da pronađe rešenje rekurzivnim pozivima funkcije `search(k+1)`. Ako se rešenje ne pronađe, kolone se otkrivaju kako bi se ispravile promene.

Iako biranje kolone sa najmanjim brojem čvorova može delovati kao najefikasnija strategija za pokrivanje u algoritmu Igrajućih Linkova, često je korisno uvesti element pseudonasumičnosti u izbor kolone. Najčešća metoda je da se, u slučaju kada više kolona ima isti minimalni broj čvorova, izabere jedna kolona nasumično. Alternativno, može se implementirati potpuna pseudonasumičnost, gde se kolone biraju prema unapred definisanoj nasumičnoj strategiji.

4 Sudoku

Sudoku je logička igra brojevima koja se igra na mreži od 9x9 kvadrata. Mreža je podeljena u devet manjih kvadrata od 3x3, zvanih "podmreže"ili "blokovi". Cilj igre je popuniti prazne kvadrate brojevima od 1 do 9, tako da svaki red, svaka kolona i svaka podmreža sadrže sve brojeve od 1 do 9

bez ponavljanja. Igra počinje sa delimično popunjenom mrežom. Brojevi u početnom rasporedu ne smeju se menjati.

Ograničenja za popunjavanje mreže:

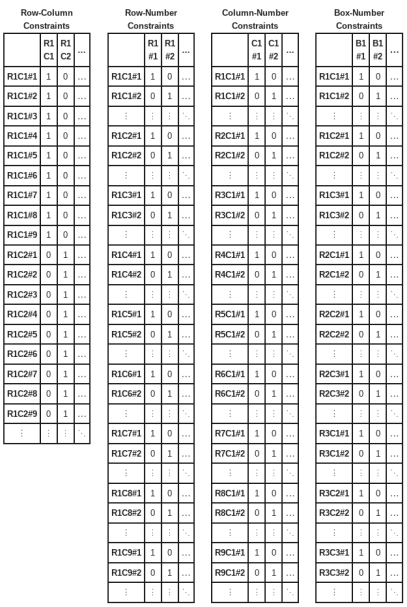
1. U svako polje se može upisati samo jedan broj
2. Svaki broj se može pojaviti samo jednom u svakom redu
3. Svaki broj se može pojaviti samo jednom u svakoj koloni
4. Svaki broj se može pojaviti samo jednom u svakoj podmreži 3x3

4.1 Svođenje na problem tačnog pokrivača

Sudoku tabla se može svesti na problem tačnog pokrivača [3]. Poenta je definisati ograničenja table kao matricu pokrivanja. Pošto u svako polje može da se upiše jedan od 9 brojeva, a tabla ima 81 polje, matrica pokrivanja ima 729 redova. Uslovi se u kolonama mogu upisati na naredni način:

1. Svaki broj od 1 do 9 može se pojaviti samo jednom u svakom polju. To zahteva 81 uslov, po jedan za svako polje, kako bi se obezbedilo da ne postoje dva broja u istom polju.
2. Svaki broj se mora pojaviti samo jednom u svakom redu. Ovo uvodi 81 uslov za sve redove.
3. Svaki broj se mora pojaviti samo jednom u svakoj koloni. Takođe, ovo uvodi 81 uslov za sve kolone.
4. Svaki broj se može pojaviti samo jednom u svakoj 3x3 podmreži. Ovo dodaje još 81 uslov za sve podmreže.

Ukupno, tablica ograničenja je dimenzije 729x324 predstavljeno kao na slici 2.



Slika 2: Definisanje uslova sudoku table preko matrice pokrivanja

5 Zaključak

Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak.

Literatura

[1] Maxime Chabert and Christine Solnon. A global constraint for the exact cover problem: Application to conceptual clustering. *Journal of Artificial Intelligence Research*, 67:509–547, 2020.

[2] Irit Dinur and Gillat Kol. Covering csps. *Weizmann Institute*, 2013.

[3] Mattias Harrysson and Hjalmar Laestander. Solving sudoku efficiently with dancing links. *Degree Project in Computer Science, DD143X*, 2014. Supervisor: Vahid Mosavat, Examiner: Örjan Ekeberg.

[4] Donald E. Knuth. Dancing links. *Millenial Perspectives in Computer Science*, pages 187–214, 2000. Comments: Abstract added by Greg Kuperberg.