Proiectarea algoritmilor

Lucrare de laborator nr. 10

Paradigma *branch-and-bound*

Perspico

Cuprins

1 Descriere

- Considerăm următoarea variantă simplificată a jocului *Perspico*.
- Fie o reţea formată din 3×3 pătrate, numite *locaţii*.
- Opt din cel nouă locaţii sunt ocupate de piese etichetate cu litere de la A la H, de dimensiuni potrivite astfel încât să poată fi deplasate pe orizontală sau verticală atunci când există o locaţie vecină liberă.
- O aşezare a piesleor în care primele opt locaţii sunt ocupate în ordinea A, B, C, D, E, F, H se numeşte *configuraţie finală* şi o notăm cu C_f (a se vedea figura 1).

A	B	C
D	E	F
G	H	

Figura 1: Jocul *Perspico*

- O *mutare* constă în deplasarea unei piese în locaţia liberă atunci când este posibil.
- Problema este următoarea:
  1. dată o configuraţie C, să se decidă dacă există o listă de mutări care să permită obţinerea lui C_f din C;
  2. în cazul în care există, să se determine o astfel listă de mutări.

2 Modelul matematic

- Spaţiul soluţiilor conţine 9! combinaţii (∞ dacă nu se verifică ciclităţile).
- În plus, interesează modul de obţinere a soluţiilor, adică drumul de la configuraţia C la configuraţia C_f.
- Dată o configuraţie C, se poate decide dacă există o listă de mutări care să permită obţinerea lui C_f din C.
- Se defineşte funcţia injectivă *poz*:

$$poz_C : \{\text{A,B,C,...,L}\} \to \{1,2,3,...,8,9\}$$

unde $poz_C(\text{X}) = i$ semnifică faptul că piesa notată cu X se află pe poziţia i. Locaţia liberă a fost asociată cu piesa simbolică L.

- În figura 2, $poz_C(\text{D}) = 2$, $poz_C(\text{L}) = 8$.

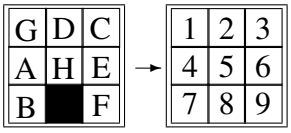

Figura 2: Configuraţia definită prin funcţia *poz*

- Pentru o configuraţie C şi pentru fiecare piesă de valoare X se defineşte

$$less_C(\text{X}) = \#\{\text{Y} | 1 \leq \text{Y} \leq \text{L}, \text{Y} < \text{X}, poz(\text{Y}) > poz(\text{X})\}.$$

- Peste mulţimea {A,B,C,...,H,L}, se consideră ordinea alfabetică.
- În exemplul anterior avem $less_C(\text{H}) = 3$, $less_C(\text{L}) = 1$.

Teorema 2.1 *Fie C o configuraţie, astfel încât $i, j \in \{1,2,3\}$ desemnează linia şi coloana unde este plasată locaţia liberă, piesa L, şi*

$$l(C) = \begin{cases} 0, & \text{dacă } i+j \text{ este par}; \\ 1, & \text{dacă } i+j \text{ este impar}. \end{cases}$$

Atunci nu există un şir de transformări până la configuraţia finală C_f dacă:

$$S(C) = \sum_{X=A}^{L} less_C(X) + l(C) \quad \text{este număr impar}.$$

3 Algoritm

3.1 Descriere

1. Presupunem că spaţiul soluţiilor candidat (fezabile) este ca şi în cazul paradigmei *backtracking*, organizat ca un arbore.
2. Rădăcina arborelui corespunde problemei iniţiale (de unde pornește căutarea), iar fiecare vârf intern corespunde unei subprobleme a problemei iniţiale.

3. Se consideră o *funcţie de predicţie* c^* definită pentru fiecare vârf v din arbore, care trebuie să satisfacă următoarele proprietăţi:

   a) $c^*(v) \leq c(v)$ pentru fiecare vârf v;

   b) dacă v este pe frontieră, atunci $c^*(v) = c(v)$;

   c) dacă w este fiu al lui v, atunci $c^*(v) \leq c^*(w)$; prin tranzitivitate, obţinem $c^*(v) \leq c^*(w)$ pentru orice descendent w al lui v.

4. Structura de aşteptare A este un min-heap pentru ca la fiecare pas să fie ales elementul cu $c^*(v)$ minim.

5. În cazul algoritmului Perspico $c^*(v) = f(v) + g^*(v)$, unde:

   - $f(v) = \text{nivel}(v)$,
   - $g^*(v)$ = numărul pieselor care nu sunt pe poziţiile definite de C_f.

3.2 Pseudocod

```
procedure perspico()
    fie r rădăcina
    calculează c*(r)
    A ← {(r, c*(r))}
    while (A ≠ ∅) do
          selectează v din A cu c*(v) minim
          A ← A \ {(v,c*(v)}
          if v este pe frontieră
             then return solutie(v)
             else fie v1,...,vk fiii viabili ai lui v
                  calculează c*(v1),...,c*(vk)
                  A ← A ∪{(vi, c*(vi)) | 1 ≤ i ≤ k}
    throw 'Nu există soluţie.'
end
```

4 Sarcini de lucru

1. Scrieţi o funcţie C/C++ care implementează algoritmul `perspico`.

2. Testați algoritmul implementat la primul punct cu diverse configurații inițiale și mărimi ale rețelei.

Bibliografie

[1] Lucanu, D. şi Craus, M., *Proiectarea algoritmilor*, Editura Polirom, 2008.