BI-ZUM - Stavový prostor a jeho prohledávání, stromová expanze, náhodné prohledávání, prohledávání do hloubky a do šířky.

Vastl Martin

May 27, 2019

Stavový prostor si lze představit jako graf, kde vrchol představuje nějaký popis stavu řešeného problému a hrany reprezentují akce a umožňují přechod mezi stavy. Příklady takových problému jsou např. hledání nejkratší cesty, symbolická integrace, a podobně. Množinu stavů značíme S a množinu akcí A.

Definice 1 *Mějme stavový prostor (S, A). Úloha prohledávání (S, A) je zadána:*

- *počátečním stavem $I \in S$,*
- *množinou koncových stavů $G \subseteq S$*

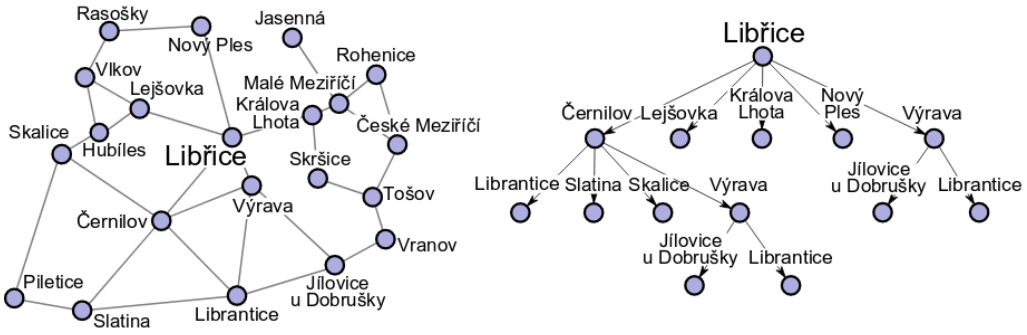
Definice 2 *Nechť (S, A) je stavový prostor a $s \in S$ je stav v tomto prostore. Řekneme, že stav $s' \in S$ je následníkem stavu s, pokud $(s, s') \in A$, tj. pokud existuje nějaká akce, která přesune stav s do stavu s'.*

Problém popsaný stavovým prostorem v praxi převádíme na jednu z následujících úloh:

- hledání cesty z počátečního do koncového stavu (nejkratší cesta, symbolická integrace i s postupem)
- hledání cílového stavu (řešení sudoku, problém n dam)

1 Stromová expanze

Definice 3 *Uvažujme stavový prostor $X = (S, A)$ počáteční stav $I \in S$ a množinu koncových stavů $G \subseteq S$. Prohledávací strom X je orientovaný kořenový strom s kořenem I, takový, každá cesta v tomto stromě se vyskytuje i v grafu (S, A).*



Prohledávání stromovou expanzí

Algoritmus prohledávání stavového prostoru stromovou expanzí lze popsat následovně:

1. Vytvoř kořen stromu coby počáteční uzel $\mathcal{I}$
   - Strom je jednoprvkový, $\mathcal{I}$ je jeho kořenem i listem
2. Dokud žádný list stromu není koncovým stavem, opakuj:
   1. „Vyber nějaký“ list stromu a prověď **expanzi**:
      - Připoj k listu všechny jeho následníky ze stavového prostoru (S, A)

Většina prohledávácích algoritmů zakazuje, aby se jeden uzel ve stromě vyskytl dvakrát. Typicky rozlišujeme 3 stavy uzlů:

- **FRESH** – uzel ještě nebyl během hledání nalezen,
- **OPEN** – uzel byl nalezen, ale ještě nebyl expandován,
- **CLOSED** – uzel byl nalezen a expandován.

Při expanzi uzlu je pak zakázáno připojení potomků, kteří jsou již ve stavu OPEN nebo CLOSED.

2 Náhodné prohledávání

Náhodně prohledávání je takové prohledávání, které náhodně otevírá jednotlivé vrcholy stavového prostoru. Je možné, že nalezne neoptimální řešení.

Algorithm 1 Random search

```
open ← {I}; closed ← {}
prev ← init_table()
while open ≠ {} do
    x ← random element of open
    if x ∈ G then
        return reconstruct_path(prev, x)
    end if
    for all y ∈ neighbors(x) do
        if y ∉ (open ∪ closed) then
            open ← open ∪ {y}
            prev[y] ← x
        end if
    end for
    open ← open \ {x}; closed ← closed ∪ {x}
end while
```

3 BFS

BFS provádí expanzi postupně po jednotlivých patrech. Zaručuje, že výsledná cesta bude mít nejmenší možný počet hran. Jeho paměťová i výpočetní čas roste exponenciálně s délkou nejkratší cesty (co do počtu hran) cesty k cíli.

Algorithm 2 Breadth-First Search (BFS)

```
open ← init_queue(); closed ← {}
prev ← init_table()
enqueue(open, I)
while ¬empty(open) do
    x ← dequeue(open)
    if x ∈ G then
        return reconstruct_path(prev, x)
    end if
    for all y ∈ neighbors(x) do
        if y ∉ (open ∪ closed) then
            enqueue(open, y); prev[y] ← x
        end if
    end for
    closed ← closed ∪ {x}
end while
```

4 DFS

DFS prochází stavový prostor a snaží se o co největší zanoření. Nemusí vždy najít nejkratší cestu, co se do počtu hran týká.

Algorithm 3 Depth-First Search (DFS)

```
1: open ← init_stack(); closed ← {}
2: prev ← init_table()
3: push(open, I)
4: while ¬empty(open) do
5:     x ← pop(open)
6:     if x ∈ G then
7:         return reconstruct_path(prev, x)
8:     end if
9:     for all y ∈ neighbors(x) do
10:        if y ∉ (open ∪ closed) then
11:            push(open, y); prev[y] ← x
12:        end if
13:    end for
14:    closed ← closed ∪ {x}
15: end while
```