Algoritmická matematika 3

Lokální vyhledávání

Petr Osička

DATA ANALYSIS AND MODELING LAB
Univerzita Palackého v Olomouci

Zimní semestr 2013

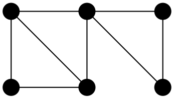
Základní idea

- technika řešení optimalizačních problémů.
- Algoritmus po celou dobu běhu udržuje **jedno aktuální přípustné řešení**.
- Na začátku si může za aktuální řešení vybrat libovolné z přípustných řešení dané instance
- Pak iterativně přechází na další přípustná řešení, které vybírá z **okolí** toho aktuálního.
- Pro potřeby algoritmu je nutné přesně určit, jak vypadá okolí libovolného přípustného řešení a také, jak z tohoto okolí jedno řešení vybrat.
- Iterace končí po splnění vhodné podmínky.

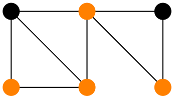
```
procedure LocalSearch(I)
    S ← InitialSolution(I)                              ▷ Vygeneruj počáteční řešení
    while True do
        C ← Neighbourhood(S, I)                         ▷ Vygeneruj okolí aktuálního řešení
        S′ ← SelectNeighbour(C, S, I)                   ▷ Najdi další řešení
        if End(I, S′, S) then                           ▷ Test konce iterace
            return S′
        end if
        S ← S′
    end while
end procedure
```

Vrcholové pokrytí

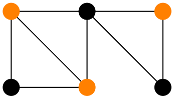
Vrcholové pokrytí grafu	
Instance:	Neorientovaný graf $G = (V, E)$.
Přípustná řešení:	$sol(G) = \{V' \mid V' \subseteq V$ a pro všechny hrany $\{u, v\} \in E$ platí, že buď $u \in V'$ nebo $v \in V'\}$
Cena řešení:	$cost(V', G) = \lvert V' \rvert$
Cíl:	minimum



(a) Neorientový graf



(b) Vrcholové pokrytí grafu



(c) Vrcholy netvořící pokrytí

Lokální vyhledávání - Vertex Cover

Za počáteční pokrytí zvolíme množinu všech uzlů.

Za okolí aktuálního pokrytí zvolíme všechna taková pokrytí, která dostaneme přídáním nebo odebráním vrcholu. Pro pokrytí C potom máme

$$\textsc{Neighbourhood}(C,(V,E)) = \{C \backslash \{v\} \mid v \in C,\ C \backslash \{v\} \text{ je pokrytí}\} \cup \{C \cup \{v\} \mid v \in E \backslash C\}.$$

Pro výběr prvku z okolí existuje několik strategií, z nichž nejjednoduší je vybrat prvek s nejlepší cenou. Této strategii se říká **gradientní metoda.**

Algoritmus končí v momentě, kdy se v okolí aktuálního řešení nevyskytuje žádné řešení s lepší cenou.

```
procedure VertexCoverGradient((V, E))
    C ← V
    F ← ∅                                         ▷ Vrcholy, ktere nelze odebrat
    while C \ F ≠ ∅ do
        Z C \ F vyber náhodný vrchol u
        x ← TRUE
        for {v, w} ∈ E do                         ▷ Pokrývá C \ {u} všechny hrany?
            if v ∉ C \ {u} and w ∉ C \ {u} then
                x ← FALSE
                F ← F ∪ {u}                       ▷ u nelze odebrat
                break
            end if
        end for
        if x then
            C ← C \ {u}                           ▷ Přejdu na další řešení
        end if
    end while
    return C
end procedure
```

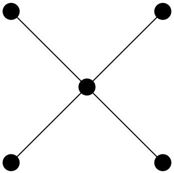
Problém uváznutí v lokálním minimu

Gradientní metoda nemusí vést k nalezení optimálního řešení.

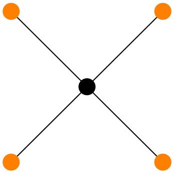
Lokální minimum
= v okolí aktuálního řešení se nenachází řešení s lepší cenou

Globální minimum
= optimální řešení

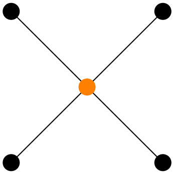
V obecném případě se lokální a globální minimum nemusí shodovat. Ukažme si několik příkladů pro VertexCoverGradient.



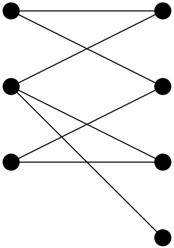
(d) Star graf



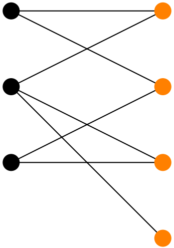
(e) Lokální minimum



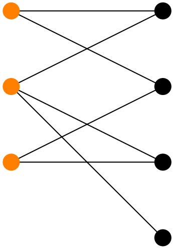
(f) Globální minimum



(g) Bipartitní graf

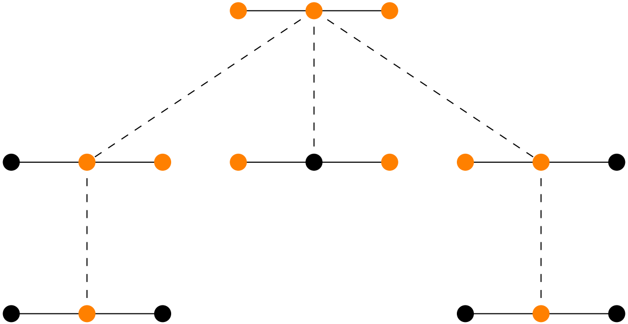


(h) Lokální minimum



(i) Globální minimum

Možné průběhy algoritmu pro malý graf



Metropolis přístup

Pokud při výběru dalších řešení z okolí aktuálního řešení umožníme zvolit i řešení s horší cenou než aktuální, můžeme někdy zabránit uváznutí tím, že „vyskočíme“ z lokálního minima (nebo z cesty do něj).

Výběr řešení z okolí vypadá následovně. Pro jednoduchost předpokládejme, že řešíme minimalizační problém.

1. Pro aktuální řešení x vybereme náhodné řešení y z okolí x (s co nejvíce uniformním rozložením pravděpodobnosti),
2. pokud $cost(x) < cost(y)$ (y je horší řešení než x), pak s pravděpodobností

$$e^{-(cost(y)-cost(x))/T}, \tag{1}$$

   kde T je konstanta, vybereme jako další řešení y (alternativou je ponechání x),
3. pokud je $cost(x) \geq cost(y)$, je dalším řešením je y.

Metropolis přístup

Poznámky k pravděpodobnosti výběru y

- Konstanta T určuje míru nestability výběru, čím je T vyšší, tím vyšší je i pravděpodobnost výběru y.
- Všimněte si také, že čím menší je rozdíl mezi cenami x a y, tím je pravděpodobnost výběru y vyšší.

Podmínky pro ukončení algoritmu

- algoritmus provede předem daný počet iterací,
- algoritmus dostatečný počet iterací za sebou nevybere jiné řešení než aktuální.

Simulované žíhání

Problém:
Metropolis algoritmu obecně trvá poměrně dlouhou dobu, než se ustálí. Důvodem je fakt, že i když se nachází blízko minima, pravděpodobnost výběru y je poměrně vysoká.

Řešení

- s rostoucím počtem iterací bude pravděpodobnost výběru y klesat.
- V počátečních iteracích existuje poměrně velká šance, že algoritmus unikne z lokálního minima se špatnou cenou
- v pozdějších iteracích už bude pravděpodobnost dostatečná malá pro to, aby se algoritmus zastavil v minimu s lepší cenou.
- Technicky to lze zajistit tak, že konstantu T nahradíme klesající funkcí závislou na počtu iterací.
- Za $T(i)$ je možné zvolit libovolnou klesající funkci takovou, aby $0 \leq e^{-1/T(i)} \leq 1$.

procedure VCANNEALING($(V, E), k$)
$C \leftarrow V$
$i \leftarrow 0$
$ch \leftarrow$ TRUE
while $i \leq k$ **do**
if ch **then**
$\mathcal{F} \leftarrow \emptyset$
for $u \in C$ **do**
$x \leftarrow$ TRUE
for $\{v, w\} \in E$ **do**
if $v \notin C \setminus \{u\}$ **and** $w \notin C \setminus \{u\}$ **then**
$x \leftarrow$ FALSE
break
end if
end for
if x **then**
$\mathcal{F} \leftarrow \mathcal{F} \cup \{C \setminus \{u\}\}$
end if
end for
for $u \in V \setminus C$ **do**
$\mathcal{F} \leftarrow \mathcal{F} \cup \{C \cup \{u\}\}$
end for
end if
Vyber náhodný prvek $S \in \mathcal{F}$
if $|S| > |C|$ **then**
S pravděpodobností $e^{-1/T(i)}$ proveď $C \leftarrow S$ a $ch \leftarrow$ TRUE
Jinak $ch \leftarrow$ FALSE
continue
end if
$i \leftarrow i + 1$
end while
return C
end procedure

Maximální řez

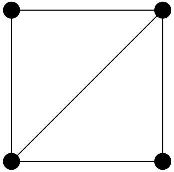
Definice

Nechť $G = (V, E)$ je neorientovaný graf a V_1, V_2 jsou disjunktní podmnožiny V takové, že $V_1 \cup V_2 = V$. Pak množinu hran $C = \{(u, v) \mid u \in V_1, v \in V_2\}$ označujeme jako **řez grafu**. Cenou řezu C je $|C|$.

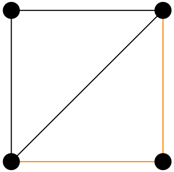
Maximální řez grafu (Max Cut)

Instance:	Neorientovaný graf $G = (V, E)$.
Přípustná řešení:	$\{C \subseteq E \mid C$ je řez grafu $G\}$
Cena řešení:	$cost(C, G) = |C|$
Cíl:	maximum

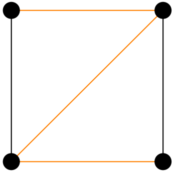
Řez grafu



(j) Ohodnocený graf



(k) Řez s cenou 2



(l) Řez s cenou 3

Maximální řez

Aproximační algoritmus, který využívá techniku lokálního vyhledávání, si v každém kroku uchovává množiny V_1 a V_2 a jim odpovídající řez C. Na začátku algoritmus zvolí tyto množiny V_1 a V_2 libovolně, například $V_1 = V$, $V_2 = \emptyset$.

Okolí aktuálního řezu C jsou všechny řezy, které vzniknou přesunem jednoho uzlu takového, že počet hran, které vedou z uzlu do množiny, ve které se nachází, je větší, než počet hran, které vedou do opačné množiny.

Z okolí pak algoritmus vybírá libovolný řez.

V pseudokódu používáme následující značení. Pro uzel v je V_v ta z množin V_1, V_2, do které tento prvek patří.

Maximální řez

procedure MaxCut$((V, E))$
    $V_1 \leftarrow V$
    $V_2 \leftarrow \emptyset$
    $x \leftarrow$TRUE
    while x **do**
        $x \leftarrow$FALSE
        for $u \in V$ **do**
            if $|\{\{u, v\} \mid V_v = V_u\}| > |\{\{u, v\} \mid V_u \neq V_u\}|$ **then**
                Přesun u mezi V_1 a V_2
                $x \leftarrow$TRUE
                break
            end if
        end for
    end while
    return C odpovídající V_1 a V_2.
end procedure