M-stromy

M-stromy jsou stromové datové struktury, které jsou podobné R-stromům a B-stromům. M-strom konstruován pomocí metriky a staví na trojúhelníkové nerovnosti pro efektivní vyhledávání rozsahu a vyhledávání k-nejbližších sousedů (k-NN). Zatímco M-stromy mohou dobře fungovat za mnoha podmínek, strom může mít také velké překrytí a neexistuje jasná strategie, jak se překrývání nejlépe vyhnout. Kromě toho je lze použít pouze pro funkce vzdálenosti, které splňují trojúhelníkové nerovnost[1].

M-strom má tyto komponenty a podkomponenty:

- **Nelistové uzly** – obsahují
  - množinu záznamů N_{RO} navigačních objektů
  - ukazatel na rodičovský objekt O_{p}.
- **Listové uzly** – obsahují
  - množinu záznamů N_{O} objektů
  - ukazatel na rodičovský objekt O_{p}.
- **Navigační objekty** – záznam navigačního objektu obsahuje:
  - (hodnoty) navigačního objektu O_{r}.
  - pokrývaný poloměr $r(O_{\mathrm{r}})$
  - ukazatel na pokrývaný strom $T(O_{\mathrm{r}})$ – všechny objekty v podstromu jsou ve vzdálenosti maximálně $r(O_{\mathrm{r}})$ od O_{r}.
  - vzdálenost $d(O_{\mathrm{r}}, P(O_{\mathrm{r}}))$ objektu O_{r} od jeho rodičovského objektu (používá se k optimalizaci při vyhledávání)
- **Objekty** – záznam objektu obsahuje:
  - (hodnoty) objektu O_j.
  - identifikátor objektu $oid(O_j)$.
  - vzdálenost $d(O_j, P(O_j))$ objektu O_j od jeho rodičovského objektu (používá se k optimalizaci při vyhledávání)

V podstatě si je můžeme představit jako R-stromy, kde nemáme MBR ale n-dimenzionální koule dané objekty a poloměrem. Tato struktura, jakožto metrický strom primárně slouží k podobnostním dotazům (nejbližší sousedi).

Hledání rozsahu

Algoritmus `Range`(Q,$r(Q)$), kde Q je objekt a $r(Q)$ je poloměr, vybere všechny objekty takové, že $d(Oj, Q) < r(Q)$. Algoritmus RS začíná od kořenového uzlu a rekurzivně prochází všechny cesty, u kterých nelze vyloučit, že vedou k objektům splňujícím tuto nerovnost.

Protože při přístupu k uzlu N je vzdálenost mezi Q a O_p (rodičovského objektu uzlu N), už vypočítané, je možné ořezat podstromu bez výpočtu vzdálenosti. Při tomto prořezávání používáme následující podmínku:

[1] mnoho pokročilých funkcí nepodobnosti použitých při načítání informací to nesplňuje

```
   def Range(N, E):
       input : N: uzel
               Q objekt vyhledávání
               r(Q) poloměr.
1      Nechť O_p je rodičovský objekt uzlu N
2      if N není list then
3          for O_r ∈ N do
4              if |d(O_p, Q) − d(O_r, O_p)| ≤ r(Q) + r(O_r) then
5                  Vypočti d(O_r, Q)
6                  if d(O_r, Q) ≤ d(Q) + r(O_r) then
7                      Range(T(O_r),Q,r(Q))
8                  end
9              end
10         end
11     else
12         for O_j ∈ N do
13             if |d(O_p, Q) − d(O_j, O_p)| ≤ r(Q) then
14                 Vypočti d(O_j, Q)
15                 if d(O_j, Q) ≤ r(Q) then
16                     Přidej O_j do výsledku
17                 end
18             end
19         end
20     end
```

Lemma 1. *Pokud $d(Q_r, Q) > r(Q) + r(O_r)$, pak pro každý objekt O_j v $T(O_r)$ platí $d(Q_j, Q) > r(Q)$. Takže $T(O_r)$ může být bezpečně ořezáno z prohledávání.*

Ovšem abychom mohli použít Lemma 1, musíme vypočítat $d(Q_r, Q)$. Tomu se dá předejít s využitím následujícího lemmatu.

Lemma 2. *Pokud $d(Q_p, Q) - d(O_r, O_p) > r(Q) + r(O_r)$, pak $d(Q_r, Q) > r(Q) + r(O_r)$.*

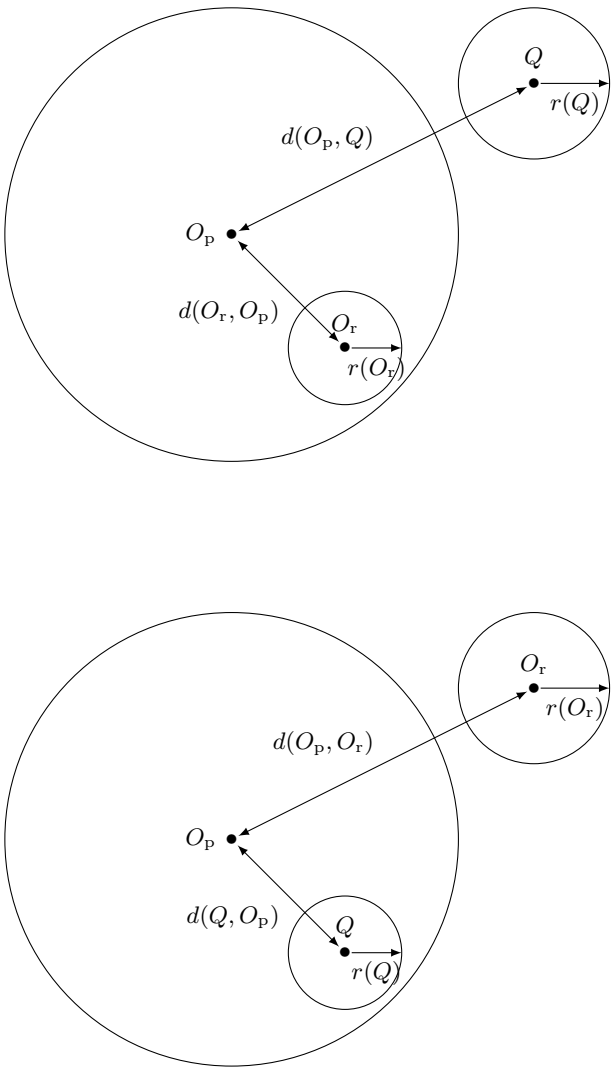
Obě ta lemmata jsou důsledky trojúhelníkové nerovnosti.

Hledání sousedů

Algoritmus `k_NN_Search` najde k nejbližších sousedů k objektu Q – předpokládá se, že minimálně k objektů je v M-stromu. Použijeme techniku branch-and-bound, celkem podobnou té, kterou jsme tu měli pr oR-stromy. Využijeme dvě globální struktury: prioritní fronta PR, a k-prvkové pole NN, které na konci provedení obsahuje výsledek.

PR je fronta ukazatelů na aktivní podstromy, tj. podstromy, které mohou obsahovat blízké sousedy. Společně s podstromy $T(O_r)$ je uložena dolní hranice $d_{\min}(T(O_r))$ vzdálenosti k objektům v $T(O_r)$. Ta dolní hranice je

$$d_{\min}(T(O_r)) = \max\{d(O_r, Q) - r(O_r), 0\}$$



Obrázek 1. Lemma 1 využito tak, abychom nemuseli počítat vzdálenosti; zjevně $d(Q_r, Q) > r(Q) + r(O_r)$ (v obou zobrazených případech), můžeme tedy ignorovat objekty O_j v podstromu $T(O_r)$.

protože žádný objekt v $T(O_r)$ nemůže mít menší vzdálenost než $d(O_r, Q) - r(O_r)$. Tyto hranice jsou používány funkcí `ChooseNode` k výběru následujícího zkoumaného uzlu z PR.

Ořezávací kritérium v `k_NN_Search` je dynamické – vyhledávaný poloměr je vzdálenost mezi Q a aktuálním k-tým nejbližším sousedem. Proto pořadí, ve kterém jsou uzly zkoumány má vliv na výkon. Heuristické kritérium využívané ve funkci `ChooseNode` je následující: vybere se ten uzel, pro který je minimální hranice $d_{\min}$ minimální.

def `ChooseNode(`PR`):`
  input : PR: prioritní fronta
1 Nechť $d_{\min}(T(O_r^*)) = \min\{d_{\min}(T(O_r))\}$ uvažujíce všechny záznamy v PR
2 Odstraň záznam $[T(O_r^*), d_{\min}(T(O_r))]$ z PR
3 vrať $T(O_r^*)$

Na konci vykonávání `k_NN_Search` bude i-tá položka pole NN mít hodnotu $NN[i] = [O_j, d(O_j, Q)]$, kde O_j je i-tý nejbližší soused Q. Vzdálenost, od i-tého záznamu bude označena d_i, takže d_k je největší vzdálenost v NN. Zjevně d_k hraje roli *dynamického poloměru vyhledávání*, protože jakýkoli podstrom splňující $d_{\min}(T(O_r)) > d_k$ může být bezpečně ořezán.

Záznamy v NN jsou iniciálně nastaveny na $NN[i] = [\_, \infty]$ $(i = 1, \ldots, k)$. Jak začne vyhledávání a přistupuje se k interním uzlům, aplikujeme následující: pro každý podstrom $T(O_r)$ vypočteme horní hranici $d_{\max T(O_r)}$ vzdálenosti objektů v $T(O_r)$ od Q. Horní hranice je

$$d_{\max T(O_r)} = d(O_r, Q) + r(O_r).$$

Uvažme nejednodušší případ, $k = 1$, dva podstromy $T(O_{r_1})$ a $T(O_{r_2})$ a předpokládejme, že $d_{\max T(O_{r_1})} = 5$ $d_{\min T(O_{r_2})} = 7$. Protože je garantováno, že $T(O_{r_1})$ obsahuje objekt, jehož vzdálenost od Q je nejvýše 5, můžeme $T(O_{r_2})$ ořezat z vyhledávání.

Následuje pseudokód `k_NN_Search`:

def `k_NN_Search(`T`,`Q`,`k`):`
  input : T – kořenový uzel
    Q – objekt, k němuž jsou vyhledáváni sousedi
    k – počet sousedů
1 $PR \leftarrow [T, \_]$
2 **for** $i \leftarrow 1$ **to** d **do**
3 $NN[i] = [\_, \infty]$
4 **end**
5 **while** $PR \neq \emptyset$ **do**
6 $NextNode \leftarrow$ `ChooseNode(`PR`)`
7 `k_NN_NodeSearch(`$NextNode$`,` Q`,` k`)`
8 **end**

Metoda `k_NN_NodeSearch` implementuje většinu logiky vyhledávání. V interním uzlu nejdříve určí aktivní podstromy, a vloží je do fronty PR. Pokud je to potřeba, zavolá funkci `NN_Update` pro vložení do NN a získání (možná nové) hodnoty d_k. Ta je pak použita k odstranění z PR těch podstromů, jejichž $d_{\min}$ je vyšší než d_k. Podobně je to provedeno pro listy. V obou případech se používá redukce výpočtů vzdálenosti využitím před-vypočítaných vzdáleností od rodičovských objektů.

```
def k_NN_NodeSearch(N,Q, k):
    input : N – uzel
            Q – objekt, k němuž jsou vyhledáváni sousedi
            k – počet sousedů
1   Nechť O_p je rodičovský objekt uzlu N
2   if N není list then
3       for O_r ∈ N do
4           if |d(O_p, Q) − d(O_r, O_p)| ≤ d_k + r(O_r) then
5               Vypočítej d(O_r, Q)
6               if d_min(T(O_r)) ≤ d_k then
7                   přidej [T(O_r), d_min(T(O_r))] do PR
8                   if d_max(T(O_r)) < d_k then
9                       d_k ← NN_Update([_, d_max(T(O_r))])
10                      Odstraň z PR všechny záznamy, pro které
                          d_min(T(O_r)) > d_k
11                  end
12              end
13          end
14      end
15  end
16  for O_j ∈ N do
17      if |d(O_p, Q) − d(O_j, O_p)| ≤ d_k then
18          Vypočítej d(O_j, Q)
19          if d(O_j, Q) ≤ d_k then
20              d_k ← NN_Update([O_j, d(O_j, Q)])
21              Odstraň z PR všechny záznamy, pro které
                  d_min(T(O_r)) > d_k
22          end
23      end
24  end
```

Vkládání

Hlavní myšlenkou je nejprve najít listový uzel N, kam nový objekt O patří. Pokud N není plný, stačí jej připojit k N. Pokud je N plný, pak vyvolat metodu spitu N.

Základní idea, použitá ke stanovení „nejvhodnějšího“ listového uzlu listu, je sestoupit v každé úrovni stromu do toho podstromu $T(O_\mathrm{r})$, pro který není nutné zvětšovat poloměr pokrytí, tj.

$$d(O_\mathrm{r}, O_n) \leq r(O_\mathrm{r}) \tag{1}$$

Pokud existuje více podstromů s touto vlastností, vybereme ten, pro který je objekt O_n nejblíže k O_r. Tato heuristika se snaží dostat dobře seskupené (shluknuté) podstromy, což je má přínosný vliv na výkon.

Pokud neexistuje navigační objekt, který by splňoval (1), vybereme ten, který minimalizuje zvětšení pokrývajícího poloměru

$$d(O_\mathrm{r}, O_n) - r(O_\mathrm{r}).$$

Tímto se snažíme minimalizovat celkový „objem“ pokrytý navigačními objekty v aktuálním uzlu.

Následuje pseudokód algoritmus vkládání:

```
   def Insert(N, E):
   |  input : N: uzel
   |           E nový záznam objektu
1  |  Nechť 𝒩 je množina záznamů v N
2  |  if N není list then
3  |  |  Nechť 𝒩_in jsou v 𝒩 ty záznamy splňující d(O_r, O_n) ≤ r(O_r)
4  |  |  if N_in ≠ ∅ then
5  |  |  |  Nechť O*_r ∈ 𝒩_in, pro který je d(O*_r, O_n) minimální
6  |  |  else
7  |  |  |  Nechť O*_r ∈ 𝒩_in, pro který je d(O*_r, O_n) − d(O*_r) minimální
8  |  |  |  r(O*_r) = d(O*_r, O_n)
9  |  |  end
10 |  |  Insert (T(O*_r),O_n)
11 |  else
12 |  |  if N není plný then
13 |  |  |  ulož O_n do N
14 |  |  else
15 |  |  |  Split(N,O_n)
16 |  |  end
17 |  end
```

Split

Jako u ostatních dynamických stromů, které jsme tu měli, i M-strom rostoe shora nahoru. S přeplnění uzlu N se vypořádáme opět tak, že vytvoříme nový uzel N' na stejné úrovni, rozdělením objektů z N mezi tyto dva uzly a posláním dvou nových navigačních objektů, které se odkazují na vzniklé uzly, do předka N_p.

```
   def Split(N, E):
       input : N: uzel
               E nový záznam objektu
1      N' ← záznamy uzlu N ∪ {E}
2      if N není kořen then
3          Nechť O_p je rodičovský objekt uzlu N, uložený v uzlu N_p
4      end
5      N' ← nový uzel
6      Promote(N',O_p1,O_p2)
7      N_1,N_1 ← Partition(N', O_p1, O_p2)
8      ulož objekty z N_1 do N
9      ulož objekty z N_2 do N'
10     if N je kořen then
11         N_p ← nový kořenový uzel
12         ulož objekty O_p1 a O_p2 do N_p
13     else
14         Nahraď objekt O_p objektem O_p1 v N_p
15         if N_p je plný then
16             Split(N_p, O_p2)
17         else
18             ulož O_p2 do N_p
19         end
20     end
```

Pokud dojde ke splitu kořene, je vytvořen nový kořen a M-strom vyroste o jednu úroveň.

Metoda `Promote` vybere a dle specifického kritéria dva navigační objekty O_{p_1} a O_{p_2}, které budou vloženy do rodičovského uzlu N_p.

Metoda `Partition` rozdělí objekty z přeplněného uzlu (množiny $\mathcal{N}$) do dvou disjunktní podmnožiny $\mathcal{N}_1$ a $\mathcal{N}_2$, které jsou pak uloženy do uzlů N a N'.

Specifické implementace metod `Promote` a `Partition` definují strategii splitu.

Bez ohledu na specifickou strategii splitu, sémantika pokrývajících poloměrů musí být zachována. Pokud je splitovaný uzel list, pokrývající poloměr povyšovaného objektu O_{p_1} je nastavena na

$$r(O_{p_1}) = \max\{d(O_j, O_{p_1}) \mid O_j \in \mathcal{N}_1\}.$$

Pokud splitujem interní uzel, pak

$$r(O_{p_1}) = \max\{d(O_r, O_{p_1}) + r(O_r) \mid O_j \in \mathcal{N}_1\}.$$

To zajišťuje, že $d(O_j, O_{p_1}) \leq r(O_{p_1})$ platí pro libovolný objekt v $T(O_{p_1})$.