Heurystyki przeszukiwania lokalnego

(c) Marcin Sydow

Zawartość wykładu

- idea przeszukiwania lokalnego
- relacja sąsiedztwa
- algorytm wspinaczkowy (hill climbing)
- algorytm symulowanego wychładzania (simulated annealing)

Heurystyki przeszukiwania lokalnego

Heurystyka (przypomnienie) jest to algorytm stosowany do rozwiązywania trudnych obliczeniowo problemów, który nie gwarantuje rozwiązania optymalnego, ale za to jest efektywny czasowo (ma akceptowalnie niską złożoność czasową)

Przeszukiwanie lokalne to zbiór heurystyk służących do rozwiązywania trudnych problemów optymalizacyjnych opartych na pomyśle lokalnego przeszukiwania przestrzeni rozwiązań.

Idea przeszukiwania lokalnego

Pomysł jest następujący:

- w przeszukiwaniu lokalnym rozpoczynamy od pewnego początkowego rozwiązania dopuszczalnego
- w każdej kolejnej iteracji przechodzimy od bieżącego rozwiązania s do innego "bliskiego" dopuszczalnego rozwiązania s', które otrzymujemy z s poprzez drobną ("lokalną") modyfikację.
- W każdej iteracji obliczamy wartość funkcji celu dla bieżącego rozwiązania, zapamiętując najlepsze.

Konkretny sposób przejścia do rozwiązania "bliskiego" w pojedynczej iteracji zależy od konkretnej wersji heurystyki.

Zalety i wady przeszukiwania lokalnego

pozytywne:

- podejście to jest uniwersalne, tzn. może być zastosowane do dowolnego problemu optymalizacji dyskretnej
- heurystyki oparte na przeszukiwaniu lokalnym są na ogół szybkie (bardziej przecyzyjnie: mają akceptowalną złożoność czasową)

negatywne:

- nie ma absolutnie żadnej gwarancji, że szybka heurystyka przeszukiwania lokalnego znajdzie rozwiązanie globalnie optymalne
- co więcej, zwykle nie ma też gwarancji, jak daleko znalezione rozwiązanie jest od globalnego optimum

Przyomnienie: graf relacji binarnej

Mając daną relację binarną $R \subseteq V \times V$ na zbiorze V, definiujemy graf odpowiadający tej relacji jako skierowany graf $G = (V, E)$, gdzie:

- zbiór wierzchołków to V
- zbiór krawędzi E to dokładnie te pary $s, s' \in V$, dla których wierzchołek s jest w relacji binarnej R z wierzchołkiem s' (czyli $(s, s') \in R$)

(przykład)

Sąsiedztwo

Dla każdego bieżącego rozwiązania (w przeszukiwaniu lokalnym) s, zbiór innych rozwiązań, które są dostępne w tej iteracji (czyli "sąsiednich") jest wyznaczony za pomocą binarnej relacji N (relacji sąsiedztwa).

Tzn. relacja sąsiedztwa $N \subseteq S \times S$ na parach rozwiązań wyznacza dokładnie te pary $(s, s') \in N$, które są "sąsiadami" (czyli z którego można przez jedną iterację przejść jako do sąsiada)

Dlatego pierwszym krokiem przy podejściu do rozwiązania problemu optymalizacyjnego za pomocą heurystyki przeszukiwania lokalnego jest *zdefiniowanie odpowiedniej relacji sąsiedztwa* na zbiorze rozwiązań.

Przykład:
Zaproponuj relację sąsiedztwa dla problemu Plecakowego

4 pożądane własności dobrze zdefiniowanej relacji sąsiedztwa

- relacja powinna być spójna (tzn. jej graf powinien być spójny). Ta cecha zapewnia, że od każdego początkowego rozwiązania można potencjalnie dotrzeć do każdego innego, w tym do rozwiązania globalnie optymalnego
- każde 2 rozwiązania będące w relacji (czyli "sąsiednie") powinny różnić się od siebie minimalnie (dzięki temu, przez drobne, szybkie "lokalne" zmiany, można przejść od sąsiada do sąsiada). Poza tym, rozwiązania sąsiednie powinny mieć podobną wartość funkcji celu.
- dla każdego rozwiązania liczba jego "sąsiadów" (wyznaczona przez relację sąsiedztwa) powinna być odpowiednio "mała" (aby efektywnie wybrać spośród wszystkich możliwych sąsiadów jednego kolejnego, co zapewnia niską złożoność obliczeniową pojedynczej iteracji)
- średnica relacji powinna być mała (mała średnica zapewnia niską maksymalną liczbę możliwych iteracji w algorytmie)

Spójność

Relacja binarna jest spójna wtedy i tylko wtedy, gdy odpowiadający jej graf jest spójny, czyli dla każdych 2 wierzchołków s i s' istnieje ścieżka w grafie łącząca te wierzchołki.

Własność ta jest niezbędna, aby zapewnić osiągalność każdego rozwiązania z każdego rozwiązania początkowego

Przykład:
W problemie plecakowym, rozwiązanie może być reprezentowane przez wektor charakterystyczny. Jak zdefiniować relację sąsiedztwa zapewniającą spójność? Jaki przykład relacji nie zapewnia spójności?

Podobieństwo sąsiadów

Jakakolwiek para sąsiadów powinna różnić się między sobą tak mało jak to możliwe, tzn. powinno być obliczeniowo łatwo (ze stałą złożonością czasową O(1), jeśli to możliwe) zmodyfikować reprezentację jakiegokolwiek rozwiązania s, aby otrzymać któregokolwiek z jego sąsiadów s'.
Dodatkowo, wartość funkcji celu dla dwóch dowolnych sąsiadów powinna się różnić tak mało jak to możliwe ("płaskość", czy "przewidywalność" wartości funkcji celu w "krajobrazie" przestrzeni rozwiązań)

Przykład:
Podać jeden dobry i jeden zły przykład relacji sąsiedztwa w problemie plecakowym zgodnie z powyższą własnością.

Małe sąsiedztwo

Relacja powinna być zdefiniowana tak, żeby dla każdego rozwiązania s liczba jego sąsiadów $|N(s)|$ nie była zbyt wysoka. Bardziej precyzyjnie: żeby była z góry ograniczona przez wielomian n, gdzie n to rozmiar zadania (np. w problemie plecakowym: liczba wszystkich przedmiotów)

Własność ta gwarantuje, że w każdej iteracji proces poszukiwania następnego sąsiada do "przejścia" jest szybki (ma co najwyżej wielomianową złożoność czasową)

Przykład: Podać jeden dobry i jeden zły przykład relacji sąsiedztwa w problemie plecakowym zgodnie z powyższą własnością.

Mała średnica relacji

Średnica relacji jest zdefiniowana jako najwyższa odległość w grafie relacji między dowolnymi dwoma wierzchołkami grafu (tu: rozwiązaniami).

Średnica relacji sąsiedztwa powinna być ograniczona z góry przez wielomian rozmiaru zadania.

Własnośc ta gwarantuje, że całkowita liczba iteracji w przeszukiwaniu lokalnym będzie ograniczona przez wielomian (co przy niskiej, wielomianowej złożoności każdej pojedynczej iteracji zapewni niską wielomianową złożoność całego algorytmu)

Przykład: Podać jeden dobry i jeden zły przykład relacji sąsiedztwa w problemie plecakowym zgodnie z powyższą własnością.

Konsekwencje własności

4 własności zapewniają:

- najlepsze globalnie rozwiązanie zawsze potencjalnie może być znalezione (dzięki spójności)
- heurystyka przeszukiwania lokalnego będzie miała akceptowalną, conajwyżej wielomianową złożoność czasową (dzięki innym własnościom)

Losowe przeszukiwanie lokalne

Najprostsze przeszukiwanie lokalne: zacząć od dowolnego dopuszczalnego rozwiązania i w kolejnych iteracjach przechodzić (przez drobną modyfikację) do kolejnego rozwiązania (sąsiedniego), dopóki zasoby czasowe na to pozwalają, rejestrując najlepsze spotkanie rozwiązanie. (zauważmy, że podejście "brute force" jest tu niemożliwe, jeśli liczba możliwych rozwiązań jest zbyt wysoka)

Istnieją doprecyzowania/ulepszenia powyższego schematu, np.:

- algorytm wspinaczkowy (hill climbing)
- algorytm symulowanego wychładzania (simulated annealing)

Algorytm wspinaczkowy (Hill Climbing (HC))

Jest to uściślenie powyższego ogólnego schematu, które mniej "ślepo" wybiera kolejny krok:

- rozpocznij od dowolnego losowego rozwiązania dopuszczalnego
- w każdej iteracji, przejdź z bieżącego rozwiązania s do tego s' z rozwiązań sąsiednich, które daje największy przyrost funkcji celu (w przypadku zadania maksymalizacji)[1] spośród wszystkich sąsiadów bieżącego rozwiązania
- kontynuuj dopóki żadne rozwiązanie sąsiadnie nie jest lepsze od bieżącego (optimum lokalne)

[1] W przypadku minimalizacji: największy spadek funkcji celu

HC: uwagi

Jest to rodzaj algorytmu "zachłannego", bo w każdej iteracji rozpatruje się tylko ruch najlepszy "lokalnie"

Przykład: Dla prawidłowo zdefiniowanej relacji sąsiedztwa w problemie plecakowym wyjaśnij do czego sprowadza się algorytm wspinaczkowy w tym przypadku. Czy w tym przypadku będzie to raczej dobry algorytm (jakie są jego wady)? Jaka jest jego złożoność czasowa?

W HC ostatnie rozwiązanie jest zawsze najlepsze spośród widzianych do tej pory.

Ulepszenia HC

Możliwe są rozmaite ulepszenia powyższego podstawowego algorytmu wspinaczkowego, np.:

- restartuj (powtarzaj) pojedynczy algorytm HC startując za każdym razem z innego początkowego rozwiązania losowego i wybierz najlepsze znalezione rozwiązanie
- przeszukiwanie wiązkowe (k-bundle search): w każdej iteracji pamiętaj równocześnie k różnych rozwiązań bieżących i rozpatruj równocześnie k najlepszych następnych ruchów. (Dlaczego jest to lepsze niż k oddzielnych powtórzeń pojedynczego HC?)

Niezależnie od powyższych 2 ulepszeń, HC zawsze idzie "do góry" (stąd nazwa)[2], co skutkuje zawsze dotarciem jedynie do najbliższego "lokalnego szczytu". (a nie globalnego)

[2] Oczywiście w przypadku zadania minimalizacji - "na dół"

Symulowane Wychładzanie (Simulated Annealing (SA))

Symulowane Wychładzanie, w przeciwieństwie do HC, pozwala w niektórych iteracjach zejść do "gorszego" sąsiada (ale prawdopodobieństwo tego spada w trakcie działania algorytmu), co sprawia, że nie jest to prosty algorytm zachłanny.

Nazwa pochodzi od analogii do procesu metalurgicznego, w którym wychładzany jest metal lub stal. Powolne (w przeciwieństwie do szybkiego) obniżanie temperatury (malejące prawdopobieństwo) sprawia, że jest większa szansa, aby zastygająca substancja znalazła lepszą (o niższej energii) konfigurację cząsteczek, bardziej przypominającą kryształ, co sprawi, że będzie mocniejsza.

Wysokopoziomowy pseudokod SA


```
current = initialRandomSolution()

while(stopCondition)
 candidate = randomNeighbour(current)
 if f(candidate) isBetterThan f(current) then
   current = candidate
 else
   current = candidate with probability
      P(current, candidate, f, temperature)
 lowerTemperature()

return(bestSeenSolution)
```

f: funkcja celu

Algorytm SA pozwala na "chwilowe" przejście do gorszego rozwiązania (aby potencjalnie "uciec" z lokalnej "pułapki"), ale z prawdopodobieństwem, które jest tym mniejsze im wyższa iteracja (wprowadza się tu pojęcie malejącej "temperatury") i im większe pogorszenie funkcji celu.

Implementacja SA: szczegóły

Konkretna wersja SA zależy m.in. od następujących decyzji:

- jaki będzie warunek zatrzymania algorytmu (np. brak ruchu przez ostatnie K iteracji (zbyt "zamrożona" struktura))
- jaki konkretnie będzie wzór na funkcję prawdopodobieństwa
- jaki będzie schemat (wzór) "zamrażania" (obniżania temperatury)

Pożądane własności funkcji prawdopobieństwa w SA

Funkcja wyrażająca prawdopodobieństwo przejścia do gorszego sąsiada w SA powinna mieć następujące własności:

- dla ustalonej pary rozwiązań (bieżące i sąsiednie) powinna maleć w czasie działania algorytmu
- dla ustalonej temperatury powinna być tym mniejsza im większe jest pogorszenie funkcji celu po przejściu do gorszego sąsiada

SA, c.d.

Przykład wzoru (funkcji) prawdopodobieństwa w SA, który spełnia powyższe własności:

$$P(curr, cand, f, temp) = e^{-(\frac{(|f(curr)-f(cand)|}{temp})}$$

W miarę jak prawdopodobieństwo spada w poszczególnych iteracjach, algorytm SA asymptotycznie zachowuje się jak HC, ale przedtem umożliwa większą eksplorację przestrzeni rozwiązań (dopóki "temperatura pozwala").

Ulepszenia SA

Możliwe są m.in. następujące ulepszenia SA:

- wariant "tabu" (niedozwolony powrót do niedawno odwiedzanych rozwiązań - aby uniemożliwić "krążenie" algorytmu)
- "tunelowanie" (dozwolone krótkie losowe "skoki" (do dalszego sąsiada), aby pozwolić potencjalnie "przeskoczyć przeszkody" w przestrzeni rozwiązań)

Przykładowe pytania/problemy

- idea przeszukiwania lokalnego
- wady i zalety przeszukiwania lokalnego
- opisz relację sąsiedztwa i wymień 4 pożądane cechy wraz z ich praktyczną rolą
- mając dany problem optymalizacji dyskretnej (np. plecakowy czy TSP) zaproponuj prawidłowo zdefiniowaną relację sąsiedztwa dla tego problemu
- mając daną relację sąsiedztwa dla danego problemu sprawdź/udowodnij czy ma pożądane 4 cechy
- napisz pseudo-kod HC i SA i wyjaśnij w języku naturalnym ich działanie, wady zalety i ulepszenia

Dziękuję za uwagę.