Algorytmy heurystyczne — 30 marca, 2011

Paweł Rzechonek *Paweł Kacprzak, Piotr Gajowiak*

1 Wstęp

Istnieją problemy, dla których nie znamy szybkich (wielomianowych) algorytmów. Co więcej, wierzymy, że takie algorytmy nie istnieją i mamy ku temu mocne przesłanki.

W naszych rozważaniach skupimy się na problemach optymalizacyjnych, czyli minimalizacji lub maksymalizacji wartość rozwiązania.

Definicja Funkcja oceny Wartość rozwiązania będziemy mierzyli funkcją określoną na przestrzeni rozwiązań, której wartości należą do zbioru liczb rzeczywistych. Taką funkcję będziemy nazywać funkcją oceny.

Definicja Rozwiązanie jako wektor Zakładamy, że rozwiązanie naszego problemu można reprezentować jako wektor. To założenie jest realne, ponieważ można je stosować do większości problemów. Przykładowo, wektorem reprezentującym rozwiązanie TSP jest sekwencja kolejno odwiedzanych wierzchołków.

Definicja Przestrzeń rozwiązań Zbiór wszystkich możliwych rozwiązań takiego problemu będziemy nazywać przestrzenią rozwiązań.

Przeszukiwanie lokalne, jak większość metod heurystycznych, jest przepisem na efektywną eksplorację tej przestrzeni, tak aby znaleźć rozwiązanie najbardziej zbliżone optymalnemu. Oczywiście, chcemy to uczynić przy niewielkim (wielomianowym względem wielkości wejścia) nakładzie czasu.

Metody przeszukiwanie przestrzeni rozwiązań możemy podzielić na dwie rozłączne klasy według metody konstrukcji ostatecznego rozwiązania:

- **Przeszukiwania zaburzeniowe -** wybieramy rozwiązanie początkowe i staramy się przez jego zaburzanie otrzymywać kolejne, lepsze rozwiązania.
- **Przeszukiwanie konstruktywne -** zaczynamy z pusty wektorem i staramy się ustawiać kolejne współrzędne tak, aby na końcu otrzymać dostatecznie dobre rozwiązanie.

Zaprezentujemy ogólną metodę przeszukiwania zaburzeniowego oraz przedstawimy rozwiązanie konkretnego problemu używając przeszukiwania lokalnego konstruktywnego

W przypadku przeszukiwania lokalnego, pomysł, a co za tym idzie, efektywność czasowa opiera się na przeszukiwania tylko części przestrzeni zwanej sąsiedztwem.

Definicja Bliskie rozwiązanie (sąsiednie) Rozwiązanie B jest bliskie rozwiązaniu A, jeśli B można otrzymać przez niewielką (elementarną) modyfikację A. Przykładowo, jeśli rozważamy metodę zaburzeniową to operacją elementarną może być modyfikacja jednej współrzędnej lub zamiana miejscami współrzędnych. Natomiast w metodzie konstruktywnej, operacją elementarną może być dodanie kolejnej współrzędnej do konstrułowanego wektora rozwiązania.

Definicja Sąsiedztwo rozwiązania Zbiór rozwiązań bliskich rozwiązaniu A będziemy nazywać sąsiedztwem rozwiązania A.

2 Rys historyczny

Trudno mówić o dacie narodzin przeszukiwania lokalnego, ponieważ, oraz posiada ona wiele modyfikacji, a główny schemat jest zbyt ogólny. Niemniej jednak, jest wiele ważnych dat związanych z tą metodą. W roku 1970 Brian Kernighan Lin oraz Lin Shen w swojej pracy zatytułowanej *An efficient heuristic procedure for partitioning graphs* zaproponowali algorytm heurystyczny, nazwany od ich nazwisk, pozwalający w czasie $O(n^2 logn)$ rozwiązywać problem podziału grafu na 2 równe części. Uznaje się ten algorytm, jako pierwszy oraz klasyczny, wykorzystujący przeszukiwanie lokalne.

W latach 90', w raz z rosnącą mocą obliczeniową, pojawiło się wielu badaczy (Selman 1992, Minton 1992, Selman 1994) zainteresowanych rozwiązywaniem problemu SAT (spełnialność formuł logicznych) przy użyciu metod przeszukiwania lokalnego. Dzięki odpowiednim metodom oraz szybkim maszynom udało sie z rozsądnym czasie otrzymywać rozwiązania dla formuł zawierających do 3000 zmiennych (wcześniej barierę stanowiły formuły z 300 zmiennymi)

Również powszechnie znana metoda heurystyczna - TS (tabu search) wynaleziona przez Freda Glovera ma za przodka przeszukiwanie lokalne.

3 Schemat główny metody

Ogólny schemat metody:

Wybierz rozwiazanie poczatkowe x i okresl jego jakość.

```
x <- najlepsze dotychczasowe rozwiazanie
x' <- x
while warunek_stopu_algorytmu:
   S <- zbiór rozwiazan z sasiedztwa x
   x'' <- najlepsze rozwiązanie z S
   x <- x''
   Jesli jakosc x jest wyzsza od jakosci x':
      x' <- x
```

Jest to najogólniejszy sposób działania przeszukiwania lokalnego. Jego siła tkwi w modyfikacjach dostosowanych do konkretnego problemu. W załączonej prezentacji przedstawiamy kilka szczegółowych schematów.

Dla przykładu, modyfikacje mogą polegać na sposobie generowania sąsiedztwa (deterministycznie lub losowo), "teleportacji" z aktualnie przeszukiwanego fragmentu przestrzenii rozwiązań do innego obszaru.

Czas działania metody zależy w głównym stopniu od sposobu generowania zbioru sąsiadów, złożoności obliczeniowej funkcji oceny i oczywiście od warunku stopu głównej pętli.

4 Prezentacja algorytmu A*

Algorytm A* jest metodą przeszukiwania grafu, wykorzystującą heurystykę w celu znalezienia najkrótszej ścieżki pomiędzy dwoma wierzchołkami.

Ścieżka ta jest budowana przed dokładanie kolejnych wierzchołków, stąd algorytm zalicza się do metod przeszukiwania lokalnego konstruktywnego.

Algorytm korzysta z funkcji heurystycznej, aby najpierw próbować budować ścieżkę z najbardziej obiecujących względem tej funkcji wierzchołków.

4.1 Schemat działania

Algorytm, zaczynając od wierzchołka startowego, w każdym kroku powiększa znalezioną aktualnie ścieżkę o wierzchołek dostępny w danym kroku oraz taki, który minimalizują funkcję kosztu dną wzorem:

$f(v) = h(v) + g(v)$

gdzie:

- h jest funkcją heurystyczną
- g jest wagą ścieżki prowadzącej od wierzchołka startowego do v (inaczej mówiąc sumą wag krawędzi na tej ścieżce)

W każdym kroku, algotym dołącza do ścieżki wierzchołek dla którego wartość f jest najmniejsza z wierzchołków osiągalnych w danym kroku.

Oczywiście funkcja heurystyczna może się mylić i konieczne mogą być nawroty.

Jednak nie są one tak straszne, ponieważ istnieją twierdzenia dowodzące optymalności algorytmu A* dla danej heurystyki, czyli nie istnieje inny algorytm, który dla tej samej heurystyki mógłby znaleźć rozwiązanie odwiedzając mniej węzłów niż A*. Dodatkowo, jeśli heurystyka jest dopuszczalna, tzn. nie zawyża rzeczywistego kosztu najkrótszej możliwej ścieżki, to mamy gwarancje, że algorytm zawsze zwróci rozwiązanie jeśli takie istnieje, a co więcej będzie ono optymalne.

Największym problemem przy zastosowaniu algorytmy dla dużych danych jest znaczne zużycie pamięci. W najgorszym przypadku A* musi zapamiętać liczbę wszystkich węzłow, która rośnie wykładniczo względem długości rozwiązania.

Istnieje jednak kilka modyfikacji algorytmu, które redukują zużycie pamięci np. kosztem czasu działania.

5 Przykład programu

Załączony do pracy, przykładowy program napisany w języku `python` z wykorzystaniem biblioteki pygame do prezentacji graficznej, rozwiązuje problem zwany "Planowaniem drogi robota" przy użyciu algorytmu A*.

Na wejściu dostajemy prostokątną plansze z polami oznaczonymi współrzędnymi, polem startowym, polem końcowym oraz listą pól zabronionych, czyli takich, na które robot nie może wejść.

Celem jest znaleznienie jak najbardziej optymalnej drogi z pola startowego do końcowego, przechodzącej tylko przez dozwolone pola.

W program zostały wbudowane 3 funkcje heurystyczne, każda z nich bierze dwa punkty przestrzeni jako argumenty i zwraca nieujemną liczbę rzeczywistą:

- **manhattan**, prosta funkcja, nazwę wzieła od mierzenia odległości między przecznicami na manhattanie. Odległość tą mierzy się tam w liniach prostych:

  `manhattan(p1, p2) = |p1.x - p2.x| + |p1.y - p2.y|`

- **diagonal**, podobna funkcja, lecz nieco o gorszych właściwościach. Działa dobrze w szczególnych przypadkach i jest szybko obliczeniowa.

  `diagonal(p1, p2) = max(|p1.x - p2.x|, |p1.y - p2.y|)`

- **euclides**, miara euklidesowa na płaszczyźnie. Ta funkcja jest dopuszczalna.

  `euclides(p1, p2) =` $\sqrt[2]{(p1.x - p2.x)^2 + (p1.y - p2.y)^2}$

Dlaczego nie stosujemy tylko funkcji euclidies? Funkcja ta potrzebuje dwukrotnego podniesienia do kwadratu oraz wyciągnięcia pierwiastka, natomiast jest dopuszczalna. Manhattan ma prostszy wzór, jednak lekko przeszacowuje. Diagonal została dodana ze względów ekperymentalnych. Jest gorsza od pozostałych funkcji.

Do programu zostały dołączone 4 przykładowe zestawy testowe.

5.1 Instrukcja uruchomienia

```
python ./prog NAZWA_FUNKCJI_HEURYSTYCZNEJ < input/testX.in
```

zamiast `NAZWA_FUNKCJI_HEURYSTYCZNEJ` wpisujemy `manhattan` lub `diagonal` lub `euclides`. `X` zastępujemy liczbą z przedziału $[1, 4]$.

Literatura

[1] K. Trojanowski, *Metaheurystyki praktycznie*, WSISiZ, 2005.

[2] W. Kernighan, S. Lin, *An Efficient Heuristic Procedure for Partitioning Graphs.* http://cm.bell-labs.com/who/bwk/partitioning.pdf

[3] P. Lester, *A* Pathfinding for Beginners,* 18 lipca 2005.
http://www.policyalmanac.org/games/aStarTutorial.htm

[4] K. Sydor, *Planowanie drogi robota, algorytm A*,* 13 maja 2008.
http://www.dioda.com.pl/forum/download.php?id=495

[5] *A* search algorithm on Wikipedia.*
http://en.wikipedia.org/wiki/A*_search_algorithm

[6] *Algorytm Kernighana-Lina w polskiej Wikipedii.*
http://pl.wikipedia.org/wiki/Algorytm_Kernighana-Lina

[7] *Tabu search on Wikipedia.*
http://en.wikipedia.org/wiki/Tabu_search