Laboratorium 6

Algorytmy optymalizacji globalnej

Wstęp

Ćwiczenie ma na celu zapoznanie się z algorytmami heurystycznymi optymalizacji globalnej. Zadaniem optymalizacyjnym jest w tym przypadku problem podziału grafu (ang. *graph partitioning*). Polega on na takim podziale zbioru wierzchołków grafu na dwa równoliczne podzbiory, by suma wag krawędzi łączących te podzbiory była jak najmniejsza. Problem ten jest NP-zupełny, więc dla dużych grafów znalezienie rozwiązania możliwie bliskiego optymalnemu wymaga zastosowania algorytmu heurystycznego. W ćwiczeniu użyte zostaną dwa takie algorytmy: Kernighana-Lina oraz algorytm oparty na symulowanym odprężaniu (ang. *simulated annealing*). Pierwszy z nich jest dedykowany do rozwiązywania problemu podziału grafu. Natomiast symulowane odprężanie jest jedną z popularnych metod metaheurystycznych ogólnego zastosowania. W ćwiczeniu badany będzie wpływ parametrów symulowanego odprężania na jakość znalezionego rozwiązania i czas działania algorytmu.

Zakładana jest znajomość:

- metody symulowanego odprężania,
- algorytmu Kernighana-Lina,
- podstawowych pojęć z dziedziny kombinatoryki i teorii grafów,
- podstaw języka C++,
- niniejszej instrukcji.

Wskazane jest także zapoznanie się z kodem programów wykorzystywanych w ćwiczeniu, zwłaszcza z treścią funkcji: *Anneal* i *KernighanLin* (plik *Optim.cpp*).

Wykorzystywane narzędzia

W ćwiczeniu będą wykorzystywane następujące programy:

- *graphgen* – generator losowych grafów nieskierowanych,
- *optim* – program dokonujący bisekcji grafu wspomnianymi dwiema metodami.

Generator grafów nieskierowanych – *graphgen*

Jest to prosty generator losowych grafów nieskierowanych. Polecenie:

```
./graphgen m n [nazwa_pliku_wynikowego]
```

spowoduje stworzenie grafu o *m* wierzchołkach i *n* gałęziach. Zostanie on zapisany w pliku o podanej na-

zwie. Jeśli nazwa zostanie pominięta, graf będzie zapisany w pliku *graf.txt*.

UWAGA! Jeśli argument *n* przekracza liczbę krawędzi pełnego grafu nieskierowanego o *m* wierzchołkach, zostanie wygenerowany właśnie graf pełny, o czym użytkownik jest informowany. Należy to brać pod uwagę podczas analizy zależności czasu wykonania algorytmów grafowych od liczby krawędzi. Natomiast próba wygenerowania grafu, w którym $n < m - 1$, zakończy się błędem, ponieważ graf taki byłby niespójny.

Generacja grafu opiera się na tworzeniu grafu pełnego i usuwaniu losowo wybranych krawędzi. Stopnie wszystkich wierzchołków grafu są w przybliżeniu równe. Wagi krawędzi to wartości bezwzględne zmiennej losowej o rozkładzie normalnym o zerowej wartości oczekiwanej, zaokrąglone w górę do najbliższej liczby naturalnej. Graf wynikowy zostaje zapisany do pliku tekstowego. Każdy wiersz takiego pliku zawiera opis jednej krawędzi:

```
nr_wierzchołka_1 <sp> nr_wierzchołka_2 <sp> waga_łuku
```

gdzie <sp> oznacza spację lub znak tabulacji. Np. wiersz o zawartości:

```
2 10 5
```

opisuje krawędź o wadze 5 wychodzącą z wierzchołka 2 i dochodzącą do wierzchołka 10. Numeracja wierzchołków rozpoczyna się od zera. Ponieważ graf jest nieskierowany, kolejność wierzchołków w opisie krawędzi nie ma znaczenia – jako pierwszy wypisywany jest wierzchołek o mniejszym indeksie.

Program optymalizujący podział grafu – *optim*

Program ten poszukuje optymalnego rozwiązania problemu podziału grafu opisanego we wstępie instrukcji. Zbiór wierzchołków grafu jest dzielony na dwie równe części, a następnie podział ten jest optymalizowany za pomocą algorytmu wykorzystującego symulowane odprężanie lub Kernighana-Lina. Elementarny krok algorytmu symulowanego odprężania polega na wybraniu po jednym wierzchołku z każdego podzbioru i zamienieniu tych wierzchołków miejscami. Program jest uruchamiany bez żadnych argumentów. Wszystkie niezbędne parametry są wczytywane z pliku *config.txt*. Oto ich lista:

GraphFile	Nazwa pliku z opisem grafu.
KernighanLinDetails	Może przyjmować dwie wartości. *T*(*ak*) powoduje, że algorytm Kernighana-Lina wyświetla w trakcie pracy szczegółowe informacje, np. o kosztach wewnętrznych i zewnętrznych wszystkich wierzchołków na danym etapie. Wartość *N*(*ie*) ogranicza te informacje do kosztu podziału po każdej iteracji, kosztu najlepszego podziału i czasu pracy.
KernighanLinRuns	Liczba uruchomień algorytmu Kernighana-Lina. Każde kolejne wywołanie rozpoczyna pracę od grafu podzielonego w poprzedniej iteracji.
ValuesFile	Nazwa pliku wynikowego, zawierającego wartości przyjmowane przez funkcję celu w kolejnych krokach algorytmu symulowanego odprężania. Domyślna nazwa to *cooling.csv*.

Format	Format pliku wynikowego o nazwie zdefiniowanej przez *ValuesFile*. Dopuszczalne wartości parametru to *matrix* i *vector* (opis w dalszej części instrukcji).
ResultsFile	Gdy symulowane odprężanie ma być przeprowadzone wielokrotnie dla różnych początkowych podziałów grafu, w pliku zapisywane są początkowe wartości funkcji celu oraz jej wartości minimalne uzyskane w każdym przebiegu algorytmu.
T	Temperatura początkowa.
p	Liczba uruchomień symulowanego odprężania. Za każdym razem losowany jest nowy początkowy podział zbioru wierzchołków grafu. Wyjątkiem jest przypadek $p = 1$ (opis w dalszej części instrukcji).
N	Liczba zmian temperatury.
s	Liczba prób (!) zamiany wierzchołków przy stałej temperaturze.

UWAGA! Jeżeli algorytm ma być uruchomiony tylko raz, początkowy podział zbioru wierzchołków na podzbiory jest dokonywany w sposób deterministyczny – na wierzchołki o indeksach parzystych i nieparzystych. Pozwala to porównać kilka wersji algorytmu (np. różniących się szybkością zmian temperatury) dla identycznego podziału początkowego.

Plik z wynikami pośrednimi, tj. plik o nazwie określonej wartością zmiennej *ValuesFile*, może mieć jeden z dwóch formatów. Format matrix ma następującą postać:

```
T1;f11;f12;f13,...
T2;f21;f22;f23,...
T2;f31;f32;f33,...
...
```

Pierwsze pole każdego wiersza to wartość temperatury, zaś pozostałe pola to wartości funkcji celu wyznaczane w tej temperaturze. Rejestrowane są oczywiście tylko te wartości funkcji celu, które albo są nie większe od wartości poprzedniej, albo zostały „zaakceptowane” przez algorytm. Pola są oddzielone średnikami. Po wczytaniu takiego pliku od arkusza kalkulacyjnego łatwo stworzyć wykres, na którym każdej temperaturze odpowiada jedna seria danych. Dzięki temu łatwo zobrazować zachowanie funkcji celu w różnych temperaturach. Format *vector* to po prostu dwie kolumny liczb: pierwsza odpowiada temperaturze, zaś druga – wartości funkcji celu w kolejnych iteracjach.

Przebieg ćwiczenia

Poniżej zostały podane przykładowe zadania. Do badań używamy grafów stworzonych przez program *graphgen*. Pracę z nim należy rozpocząć od nadania stałej SEED wartości swojego numeru albumu. **Sugeru-**

je się generowanie grafów o kilkudziesięciu wierzchołkach i kilku-kilkunastokrotnie większej liczbie krawędzi. **Im większy graf, tym mniej zróżnicowane są koszty wszystkich możliwych podziałów. Dla grafów o kilkunastu lub kilkudziesięciu tysiącach krawędzi możliwy do osiągnięcia zysk z zastosowania algorytmów optymalizacyjnych jest znikomy.**

1. Dla grafu o kilkudziesięciu wierzchołkach i stu kilkudziesięciu (ewentualnie kilkuset) krawędziach przeprowadzamy jednokrotnie (p = 1) symulowane odprężanie dla temperatury zbliżonej do zera. **UWAGA!** Konstrukcja programu *Optim* nie pozwala na podanie w pliku *config.txt* wartości parametru T równej zeru. Jeśli zamierzamy przeprowadzić eksperyment przy „zerowej” temperaturze, parametr T powinien być b. małą liczbą dodatnią, np. 1e-6. Następnie należy odpowiedzieć na następujące pytania:
   a) Co można powiedzieć o symulowanym odprężaniu prowadzonym temperaturze bliskiej zeru? Do jakiego algorytmu upodabnia się ono w takich warunkach?
   b) Po ilu krokach algorytm znalazł minimum funkcji celu? Informację tę można znaleźć w pliku z wartościami przyjmowanymi przez funkcję celu w kolejnych krokach (jego domyślna nazwa to *cooling*.csv). **Notujemy** liczbę kroków oraz wartość znalezionego minimum funkcji celu.
   c) Czy zwiększenie liczby kroków (tj. zwiększenie wartości N lub s) miałoby sens?

   Następnie zwiększamy temperaturę do wartości z przedziału 0.1–2. Uruchamiamy program *Optim* dla różnych wartości parametrów N i s. Sugerowane wartości N to kilkaset do kilku tysięcy, zaś s – kilkadziesiąt do kilkuset, choć oczywiście można pokusić się o użycie większych wartości.
   d) Dla każdej kombinacji wartości N, s zapisujemy wartość znalezionego minimum funkcji celu.
   e) Ile (w przybliżeniu) wynosi teraz wartość funkcji celu po takiej liczbie kroków, po jakiej algorytm znalazłby minimum przy $T \approx 0$ (patrz podpunkt b)?
   f) Co zyskujemy, a co tracimy przyjmując $T \approx 0$?
   g) Na podstawie zawartości pliku *cooling.csv* sporządzamy wykres, na którym nanosimy wartości funkcji celu w kolejnych krokach przy $T \approx 0$ oraz przy $T > 0$ (dla wybranej kombinacji parametrów s i N). Wykres ma uzasadniać wnioski wyciągnięte w punkcie f). **UWAGA!** Wszelkie wykresy dla dużych ilości danych wklejamy do sprawozdania po konwersji na rysunek (.tiff, .png itp.), **nie** przenosimy z arkusza kalkulacyjnego metodą „kopiuj i wklej”.
2. Dla grafu o kilkudziesięciu/kilkuset wierzchołkach i odpowiedniej liczbie krawędzi uruchamiamy algorytm symulowanego odprężania kilkadziesiąt lub kilkaset razy (nadając odpowiednią wartość parametrowi p w pliku konfiguracyjnym). Za każdym razem algorytm rozpoczyna pracę od innego wstępnego podziału grafu. Na podstawie wyników zapisanych w pliku *costs.txt*, czyli początkowych i najlepszych kosztów podziału w każdym z p uruchomień:
   a) Notujemy najmniejszą i największą uzyskaną wartość funkcji celu spośród wszystkich p „zoptymalizowanych” wartości.

b) Zarówno dla podziałów początkowych, jak i dla podziałów zoptymalizowanych, wykreślamy histogramy wartości funkcji celu lub skumulowane funkcje prawdopodobieństwa, czyli dystrybuanty empiryczne. W tym drugim przypadku argumentami są **posortowane** wartości wczytane z pliku *costs.txt*, zaś wartościami – kolejne liczby: 1/*p*, 2/*p*, ..., 1, gdzie *p* jest liczbą uruchomień algorytmu. Jakie wnioski można wyciągnąć na podstawie porównania histogramów (lub dystrybuant) dla podziałów początkowych i wynikających z optymalizacji?

3. Załóżmy, że dysponujemy ograniczonym czasem na znalezienie możliwie dobrego rozwiązania problemu podziału. Innymi słowy, mamy do dyspozycji jedynie ograniczoną liczbę kroków *k*, którą możemy przedstawić jako następujący iloczyn:

$$k = s \cdot N \cdot \mathrm{p},$$

gdzie: *s* – liczba prób przy stałej temperaturze, *N* – liczba zmian temperatury, *p* – liczba różnych podziałów początkowych, dla których uruchamiamy algorytm. Dla odpowiednio dużego grafu (kilka/kilkadziesiąt tysięcy krawędzi) i odpowiednio dużego *k* zmieniamy wartości parametrów *s*, *N* i *p* tak, by utrzymać stałą wartość *k*. Dla każdej kombinacji należy wykonać kilka eksperymentów i wybrać najlepszy wynik. Wartości *s*, *N*, *p*, koszt początkowy i minimalny należy przedstawić w tabeli i skomentować. Czy jakaś kombinacja tych trzech parametrów okazuje się wyraźnie korzystniejsza od innych ze względu na jakość ostatecznego rozwiązania lub szybkość jego osiągnięcia? Jakie ogólne wnioski można wysnuć na tej podstawie?

4. Modyfikujemy funkcję odpowiadającą za zmiany temperatury w kolejnych krokach. Eksperymentujemy z różnymi grafami, wartościami temperatury początkowej, liczbami kroków itp. Czy któraś z wersji daje konsekwentnie lepsze wyniki od innych? Przykładowe wyniki odczytane z pliku *cooling.csv* (tj. wartości temperatury i funkcji celu w kolejnych krokach) przedstawiamy na wykresach na poparcie wniosków. Przykładowe modyfikacje:

   a) Zmiana wartości współczynnika *a*, decydującego o szybkości zmian temperatury.

   b) Zmiana scenariusza schładzania. Zamiast używanej w programie krzywej wypukłej:

$$T(n+1) = aT(n)$$

   stosujemy liniowy spadek do wartości bliskiej zeru – funkcja *cool2*(). Można również poeksperymentować z własnymi przebiegami temperatury w funkcji czasu. Parametry dobieramy tak, by wszystkie funkcje osiągały wartość bliską zeru po podobnej liczbie kroków *N*. Zmodyfikowany kod, wyniki eksperymentów i wnioski zamieszczamy w sprawozdaniu.

5. W miarę działania algorytmu prawdopodobieństwo wykonania akceptowalnej zamiany wierzchołków maleje, gdyż większość posunięć zmniejszających wartość funkcji celu zostało już wykonanych, a prawdopodobieństwo zaakceptowania posunięcia zwiększającego wartość funkcji celu maleje w miarę zmniejszania temperatury. Aby nie dopuścić do zbyt szybkiego utknięcia w minimum lokalnym, można:

a) uzależnić liczbę prób wykonywanych przy danej temperaturze (zmienna *s*) od temperatury – w niższych temperaturach zmienna *s* powinna przyjmować większe wartości,

b) skokowo zwiększyć temperaturę, gdy liczba nieudanych prób zamiany wierzchołków przekroczy pewną wartość; jeśli dzięki temu posunięciu uda się wyjść z minimum lokalnego, można wrócić do „starej” temperatury lub stopniowo zmniejszać ją według dotychczasowego scenariusza.

c) wprowadzić inną sensowną adaptację.

Należy zaobserwować, jak wprowadzenie tych modyfikacji wpływa na czas wykonywanych obliczeń oraz na uzyskiwane wyniki. W sprawozdaniu zamieszczamy:

- opis wprowadzonych zmian (najlepiej poparty odpowiednimi fragmentami kodu),
- opis eksperymentów, które pozwoliły porównać wydajność zmodyfikowanego algorytmu z oryginałem – wartości użytych parametrów (*N*, *s* i ewentualnie własnych parametrów),
- wyniki – osiągnięte wartości minimalne funkcji celu, liczbę kroków potrzebną do osiągnięcia tych minimalnych wartości, przykładowe wykresy itp.,
- wnioski.

6. Dla grafu o kilkudziesięciu/kilkuset wierzchołkach i odpowiedniej liczbie krawędzi dokonujemy podziału algorytmem Kernighana-Lina a następnie algorytmem symulowanego odprężania z takimi wartościami parametrów (*p*, *N*, *s*, temperatura, scenariusz schładzania, ewentualne własne adaptacje itp.), jakie wydają nam się najlepsze. Następnie:

   a) Wykreślamy wartość funkcji celu po kolejnych krokach algorytmu Kernighana-Lina. Ile było kroków? W którym kroku zostało wyznaczone optimum? Czy pracę algorytmu można było przerwać w momencie, gdy wartość funkcji celu zaczęła rosnąć?

   b) Porównujemy wartość funkcji celu w optimach znalezionych przez oba algorytmy oraz czas pracy każdego z nich. Co można powiedzieć o obu algorytmach?

   c) Graf po podziale algorytmem Kernighana-Lina poddajemy dalszej optymalizacji – tym samym algorytmem lub metodą symulowanego odprężania. Czy funkcja celu uległa poprawie?

Ostatnia modyfikacja 26.06.2017

Opracował dr inż. Dominik Kasprowicz

Konsultacja merytoryczna: dr inż. Adam Wojtasik