Maurycy Niewczas nr. 252814

# Projektowanie Efektywnych Algorytmów

# Projekt 2

W ramach projektu dokonałem implementacji oraz analizę efektywności algorytmu **Symulowanego Wyżarzania (Simulated Annealing)** dla asymetrycznego problemu komiwojażera (ATSP).

1. Wstęp teoretyczny

Algorytm Symulowanego Wyżarzania został po raz pierwszy opisany w 1953r. przez Metropolisa. Swoją nazwę i sposób działania zawdzięcza analogi do zjawisk fizycznych jakim jest proces ochładzania i stygnięcia metalu. Algorytm polega na ciągłym poprawianiu i ulepszaniu bieżącego rozwiązania. Potrafi on wychodzić z minimum lokalnego, w celu optymalizacji w kierunku minimum globalnego. Dodatkowo zaimplementowałem zapamiętywanie dotychczasowej ścieżki o najmniejszym koszcie co sprawia, że wyniki są bliższe optymalnemu rozwiązaniu.

2. Założenia projektu

Do obliczenie początkowej temperatury wykorzystuję koszt początkowej ścieżki pomnożony przez liczbę wierzchołków. Długość epoki to liczba wierzchołków pomnożona przez 20. Współczynnik redukowania temperatury α=0,95. Chłodzenie temperatury odbywa się na podstawie mnożenia temperatury przez współczynnik α. Jest to tak zwane chłodzenie geometryczne. Do pomiaru czasu przyjąłem generowanie 10 losowych instancji (wynik nie wpływa znacząco na średnią) dla każdej wielkości N={8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40}. Do sprawdzenie błędu wygenerowanego kosztu symulowanego wyżarzania porównuje każdą instancje z algorytmem branch and bound.

3. Sposób działania algorytmu

W programie użyłem wskaźników do dynamicznego stworzenia tablic. Metoda shuffle losuje pierwszą losową permutacje. Metodą neigbourPermutation wyznaczam losowo sąsiada bieżącej permutacji sposobem swap. Następnie sprawdzam różnicę pomiędzy bieżącym a sąsiednim rozwiązaniem. Kiedy ta różnica jest mniejsza przypisuję do bieżącego rozwiązania, odpowiednie rozwiązanie sąsiednie. Zawsze w przejściu pętli porównuję najmniejsze dotychczasowe znalezione rozwiązanie z rozwiązaniem bieżącym. Robię tak dlatego by rozwiązanie końcowe było bliżej rozwiązaniu optymalnemu. Sposobem wyjścia z minimum lokalnego stosuje odpowiednie porównanie przy każdym przejściu pętli: $x < \exp(\frac{-\delta}{\mathrm{t}})$

Gdzie:

x- losowa liczba zmiennoprzecinkowa z przedziału (0;1)

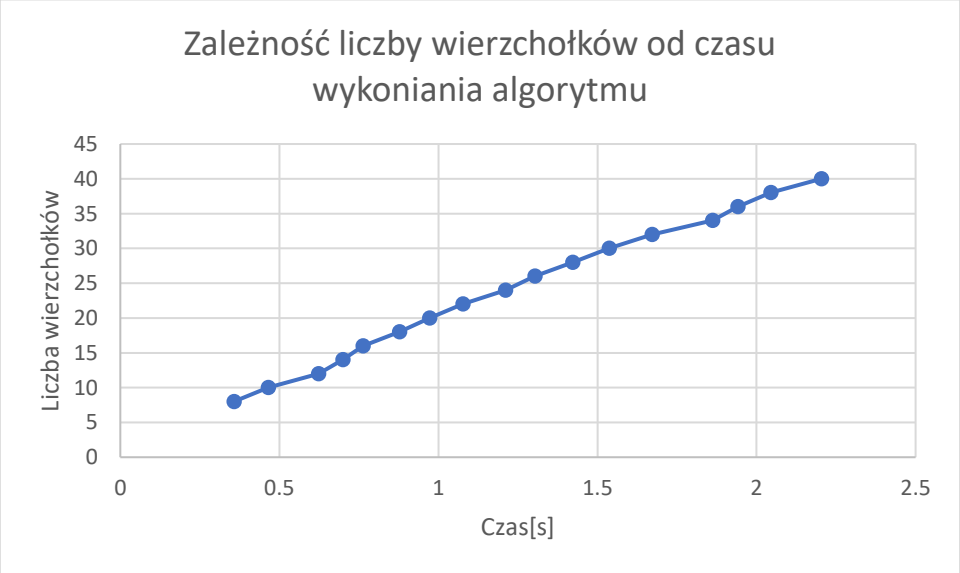
δ- różnica rozwiązania bieżącego z sąsiednim

t- temperatura

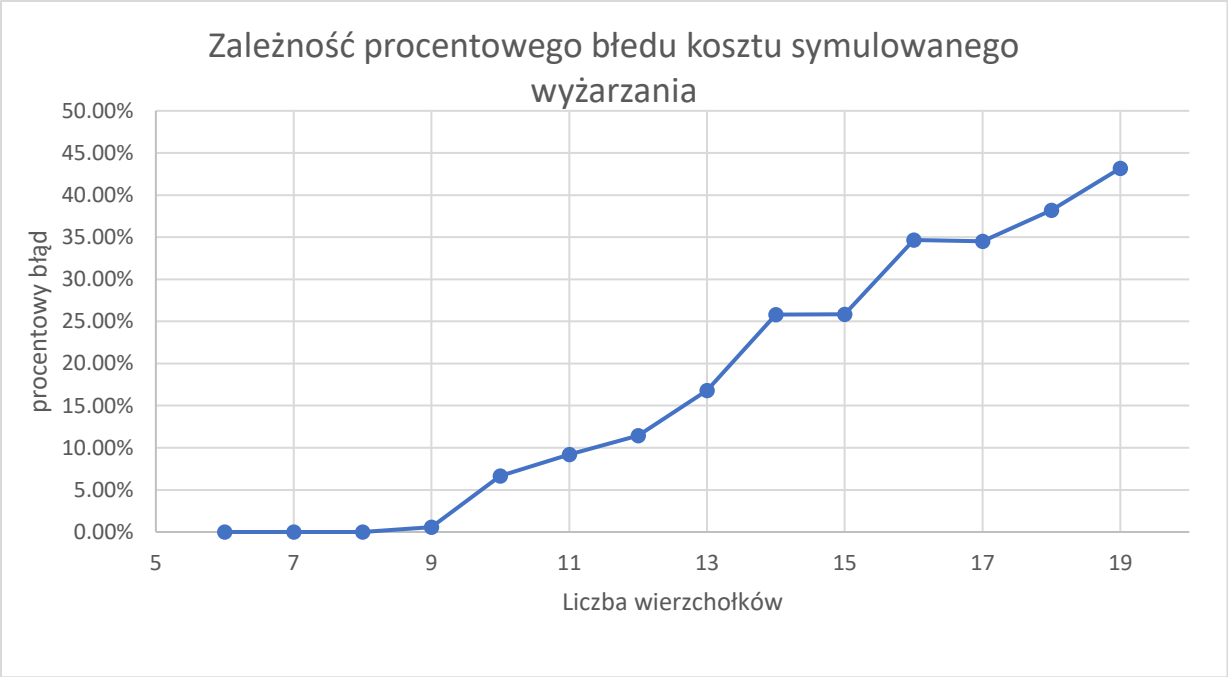
Następnie po wyjściu z pętli wewnętrznej temperatura jest chłodzona geometrycznie

4. Wyniki pomiarów

| liczba wierzchołków | średnia czasu[s] |
|---|---|
| 8 | 0,357183 |
| 10 | 0,465318 |
| 12 | 0,623412 |
| 14 | 0,699453 |
| 16 | 0,76294 |
| 18 | 0,877742 |
| 20 | 0,971777 |
| 22 | 1,076984 |
| 24 | 1,210642 |
| 26 | 1,303475 |
| 28 | 1,422006 |
| 30 | 1,536741 |
| 32 | 1,671748 |
| 34 | 1,861372 |
| 36 | 1,941164 |
| 38 | 2,044599 |
| 40 | 2,203609 |




Parametry; temperatura początkowa, długość epoki, współczynnik chłodzenia alpha mają wpływ na czas wykonania algorytmu. Przy sztywnym wpisaniu temperatury początkowej i długości epoki, czas działania algorytmu byłby zbliżony, ponieważ od tych parametrów zależy ilość wywołań pętli.




Jak widać na powyższym wykresie błąd procentowy otrzymanego kosztu rośnie wraz z większą liczbą wierzchołków.

5. Przykład praktyczny

Dla grafu o 5 wierzchołkach parametry zostały ustawione następująco

```
wykonaj
dlugosc epoki L:         100
Temperatura poczatkowa: 1050
wspolczynnik alpha:      0.95
```

Przy czym alpha zawsze przyjmuje domyślnie 0,95, natomiast jej wartość może być zmieniona w programie. Pierwsza ścieżka jest ustawiana jako 0, 1, 2, 3, 4, lecz w następnym kroku wartości są losowo wymieszane w tablicy, w tym przypadku: 0, 2, 3, 4, 1. Pierwszy wierzchołek „0” nie może zmienić swojej pozycji w tablicy. Jest to uwzględnione także przy generowaniu tablicy sąsiedztwa. Minimalny koszt oraz ścieżka są na początku przypisywane do ścieżki początkowej oraz kosztu początkowego. W pętli wewnętrznej przez długość epoki generowani są sąsiedzi. Każdy sąsiad jest porównywany kosztem z aktualną otrzymaną ścieżką. Gdy sąsiad jest tańszy rozwiązaniem staje się aktualnym rozwiązaniem, dla którego w dalszych wywołaniach będą generowani sąsiedzi. Tablica z minimalnym rozwiązaniem po prostu przechowuje dotychczasowo znalezione najmniejszą ścieżkę.

```
146             //ustawianie minimalnego kosztu na wypadek wyjscia z minimum lokalnego
147             if (neigCost < minCost)
148             {
149                 minCost = neigCost;
150                 copy(neighbour, minPath);
151             }
152             if (costDiff < 0)
153             {
154                 currentCost = neigCost;
155                 copy(neighbour, path);
156             }
157             else
158             {
159                 x = g.generateRandomFloat(0, 1);
160                 if (x < pow(EULER,((float)-costDiff/(float)temp)))
161                 {
162                     currentCost = neigCost;
163                     copy(neighbour, path);
164                 }
165 
166             }
```

Do wyjścia z minimum lokalnego stosuję się generowanie losowej liczby z przedziału (0;1) oraz przystawieniu jej jako x do wzoru $x < \exp(\frac{-\delta}{t})$ co obrazują linijki od 157 do 164. Na końcu jako wynik algorytmu zwracany jest dotychczasowy najmniejszy koszt oraz przyporządkowana do tego kosztu ścieżka.

6. Wnioski

Algorytm Symulowanego Wyżarzania radzi sobie z taką ilością wierzchołków z jakimi nie radzą sobie algorytmy b&b i Brute Force. Czas oczekiwania tych dwóch jest znacznie większy niż symulowanego wyżarzania dla większej ilości wierzchołków. Jednak dzieje się to kosztem wytworzenia rozwiązania zbliżonego a nie rozwiązania optymalnego.