Projektowanie Efektywnych Algorytmów
Projekt

22/12/2022

259193 Kacper Wróblewski

(4) Algorytm symulowanego wyżarzania

Spis treści	*strona*

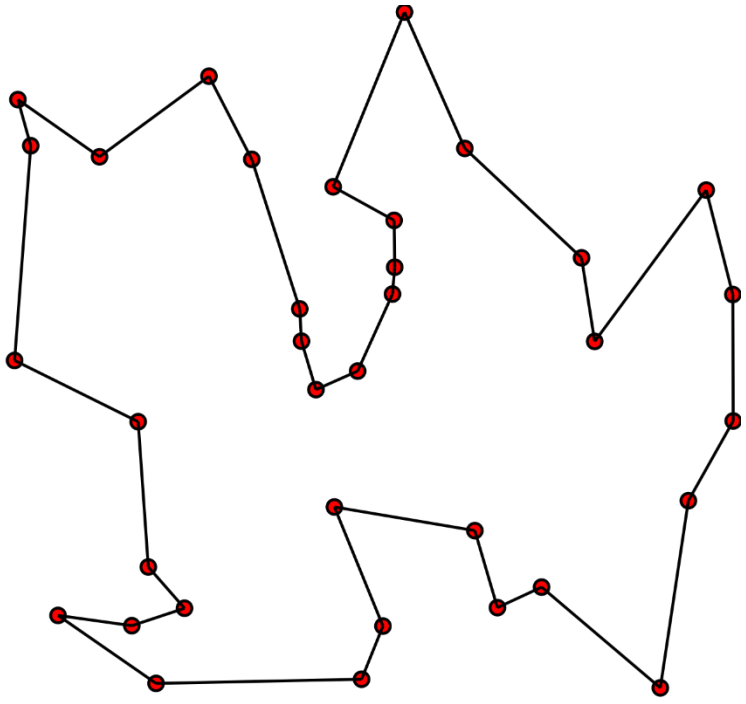
1. Sformułowanie zadania

Zadanie polega na opracowaniu, implementacji i zbadaniu efektywności algorytmu symulowanego wyżarzania rozwiązującego problem komiwojażera w wersji optymalizacyjnej. Algorytm był realizowany na gotowym grafie stworzonym z danych do opracowania. Ze względu na specyfikę metody wśród danych do opracowania zamieszczono również dokładny wynik najkrótszej drogi dla każdej instancji jako punkt orientacyjny dla wyliczania wartości błędu.

Należało zbadać zależność czasową od wielkości instancji (jak w zadaniu 1. i 2.) oraz pamięciową (jak w zadaniu 2.).

Problem komiwojażera, czyli TSP (*travelling salesman problem*) polega na znalezieniu cyklu Hamiltona w grafie, który ma najmniejszy koszt. Sprowadza się to do wyznaczenia najkrótszej ścieżki pomiędzy wierzchołkami przedstawianymi jako miasta, stąd problem podróżującego sprzedawcy. Dana jest określona ilość miast i odległość albo cena podróży pomiędzy nimi. Podróżujący musi odwiedzić wszystkie z nich płacąc jak najmniej lub podróżując jak najmniejszą odległość. Jak przedstawiono na Rys. 1., rozwiązaniem jest cykl w grafie zupełnym w postaci listy kolejnych wierzchołków oraz całkowity koszt przebytej drogi.

Problem TSP należy do klasy problemów NP-trudnych, co znaczy, że rozwiązanie nie zawsze jest jasne lub zajmuje zwyczajnie zbyt długo, aby brać je pod uwagę przez co wymagane jest częste zawężanie kryteriów lub kontemplacja jakości algorytmu w danym kontekście.



https://pl.wikipedia.org/wiki/Problem_komiwojażera#/media/Plik:GLPK_solution_of_a_travelling_salesman_problem.svg

Rys. 1. Przykładowe rozwiązanie problemu TSP

2. Metoda

Symulowane wyżarzanie jest to metoda projektowania algorytmów heurystycznych, to znaczy takich, które nie dają gwarancji znalezienia rozwiązania optymalnego. Nie jest to jednak problem, ponieważ dzięki specyfice tej metody zadowalające wyniki otrzymywane są w rozsądnym czasie.

Algorytmy heurystyczne wykorzystuje się, gdy otrzymywane optymalne rozwiązania mają zbyt dużą złożoność obliczeniową lub nie są możliwe do znalezienia. Opisywany problem komiwojażera klasyfikuje się jako problem NP-trudny, ergo wykorzystanie metody symulowanego wyżarzania jest usprawiedliwione.

Symulowane wyżarzanie swoją nazwę zawdzięcza procesowi wykorzystywanemu w metalurgii, w którym stopniowo schładza się metal, stąd jednym z głównych parametrów algorytmu jest temperatura początkowa. Technika opiera się na wylosowaniu pewnego rozwiązania z puli dostępnych, a następnie iteracyjne modyfikowanie go celem otrzymania rozwiązania bardziej optymalnego. Jeżeli po wylosowaniu rozwiązania, w kolejnym kroku otrzyma się lepsze – po prostu wybierane jest lepsze. Istnieje jednak możliwość zaakceptowania gorszego rozwiązania z pewnym prawdopodobieństwem, ma to na celu umożliwienie ucieczki z minimum lokalnego funkcji określającej przestrzeń rozwiązań.

Prawdopodobieństwo przyjęcia gorszego rozwiązania zawiera się we wzorze prezentującym rozkład Boltzmanna:

$$e^{\frac{f(X)-f(X')}{T}}$$

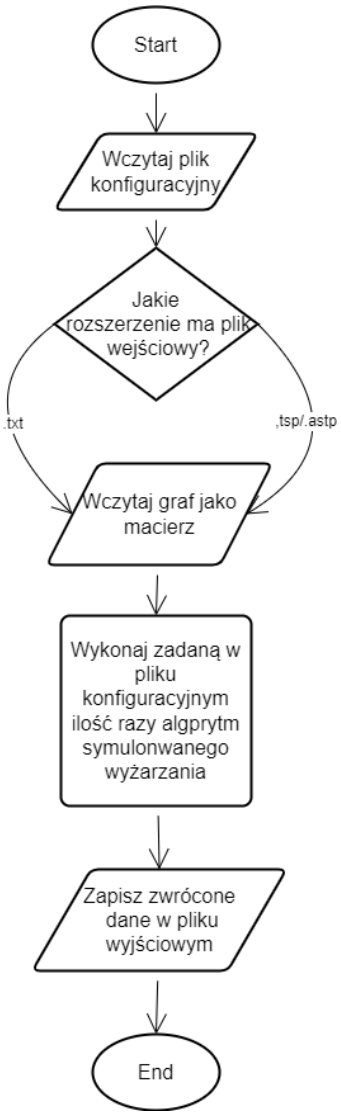
X – poprzednie rozwiązanie,

X' – nowe rozwiązanie

f – funkcja jakości (im większe X, tym lepsze rozwiązanie, w tym przypadku jest to koszt ścieżki)

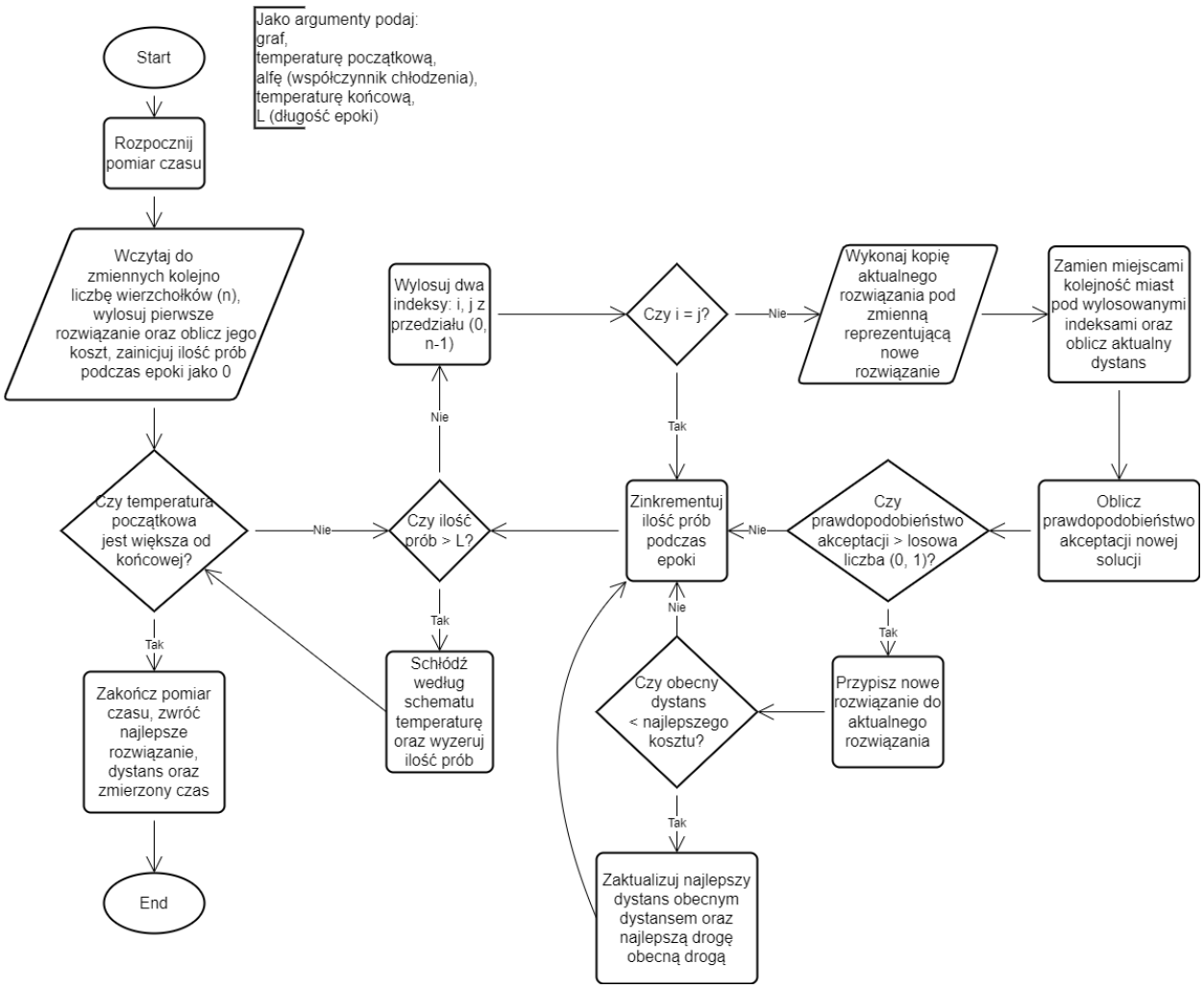
Oprócz temperatury początkowej metoda ta przyjmuje inne parametry, takie jak sposób obniżania temperatury (współczynnik schładzania) oznaczany jako α. Zazwyczaj jest to liczba bliska 1. Oprócz tego, określa się liczbę przeprowadzonych prób w trakcie tej samej temperatury (czas trwania epoki) – L. W algorytmie podaje się również warunek stopu, czyli co musi się stać, aby można było zmniejszyć temperaturę (może to po prostu być wykorzystanie zadanej liczby prób w danej epoce). Algorytm kończy swoje działanie bez względu na jakoś rozwiązania po otrzymaniu temperatury końcowej.

3. Algorytm



Rys. 2. Schemat blokowy całego programu

Program, jak przedstawiono na Rys. 2., rozpoczyna się od wczytania danych zawartych w pliku konfiguracyjnym. Potem należy rozpatrzeć przypadek rozszerzenia pliku, gdyż część z badanych zbiorów ma inny format niż *.txt*, który wymaga nieco innej metody odczytu (pliki *.tsp/.astp* można łatwo odczytać za pomocą biblioteki *tsplib95* w pythonie). Następnie program wchodzi w pętlę realizującą zaimplementowany algorytm. Liczba iteracji tej pętli określona jest w pliku konfiguracyjnym. Gdy program zakończy iterację, zapisuje zwrócone dane do pliku, którego ścieżka i nazwa również znajdują się w pliku sterującym, czym powinien zakończyć się program. Opcjonalnie, na końcu programu można wypisać w konsoli interesujące programistę dane, w celu szybszej weryfikacji poprawności metody.



Rys. 3. Schemat blokowy implementacji algorytmu symulowanego wyżarzania

Implementacja algorytmu symulowanego wyżarzania rozpoczyna się startem pomiaru czasu.

Następnie deklarowane są wszystkie potrzebne zmienne oraz kolekcje do zapamiętania wyników, tj. *n* – liczba miast, losowane jest aktualne rozwiązanie (i od razu przypisywane jako najlepsze), wyliczany jest najlepszy koszt (pochodzący z aktualnego rozwiązania) oraz deklarowana jest ilość prób jako 0 (opisane przy polu *start* na rys. 3).

W kolejnym kroku program wchodzi w zewnętrzną pętlę, która będzie się wykonywać dopóki nie zostanie osiągnięta temperatura końcowa.

Potem program wchodzi w zagnieżdżoną pętlę ograniczoną warunkiem ilości prób w epoce. Następnie losowane są dwie zmienne stałoprzecinkowe, *i* oraz *j* z przedziału (0, *n*-1). Posłużą one do zamiany dwóch miast miejscami w rozwiązaniu, o ile nie wylosują się takie same.

Jeśli ten warunek jest spełniony, to do zmiennej przechowującej nowe rozwiązanie kopiowana jest lista w aktualnej kolejności wierzchołków, gdzie miasta o indeksach *i* oraz *j* są zamieniane miejscami celem

optymalizacji rozwiązania. Po zamianie wyliczany jest nowy oraz aktualny dystans. Te informacje służą obliczeniu prawdopodobieństwa akceptacji rozwiązania, którego wzór został podany w punkcie 2.

Jeżeli prawdopodobieństwo akceptacji jest większe od wylosowanej liczby z przedziału (0, 1) to rozwiązanie jest aktualizowane listą z zamienionymi miastami. Następnie, jeżeli obecny dystans jest lepszy niż poprzedni najlepszy to zmienne przechowujące optymalne rozwiązanie (ścieżkę oraz koszt) są aktualizowane tymi obecnymi.

Po wyjściu z tych instrukcji warunkowych inkrementowana jest liczba prób celem przejścia do następnej liczby epoki, czym kończy się ciało zagnieżdżonej pętli.

Na zakończenie epoki zerowana jest liczba prób oraz schładzana jest temperatura, po czym program wchodzi w kolejną iterację pętli, lub jeśli warunki podane wyżej nie są spełnione, kończy pomiar czasu oraz zwraca go wraz z najkrótszą wyliczoną ścieżką oraz jej kosztem.

4. Dane testowe

Do sprawdzenia poprawności działania algorytmu i przeprowadzenia badań wybrano następujący zestaw instancji (wyniki zebrano w punkcie 6.):

http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

Plik:	*Optymalne wartości (koszt):*
1. *gr96.tsp*	*55209*
2. *kroa100.tsp*	*21282*
3. *krob150.tsp*	*26130*
4. *krob200.tsp*	*29437*
5. *pr152.tsp*	*736382*
6. *ftv170.atsp*	*2755*
7. *rbg323.atsp*	*1326*

Pliki testowe są w następującym formacie:

Pliki sformatowano do *.txt* oraz dodano optymalną drogę w pierwszej linii celem wczytania do programu danych potrzebnych do wyliczenia błędu pomiaru. Kolejną linię zajmuje wielkość grafu. W kolejnych znajduje się macierz sąsiedztwa.

Do programu został dołączony plik *config.ini*, aby sterować parametrami programu w sposób zastępujący:

```
Plik wejściowy:
./input/gr96.txt                    //ścieżka pliku wejściowego
Plik wyjściowy:
./output/data_instances.csv         //ścieżka pliku wyjściowego
Ilość powtórzeń:
1                                   //określenie, ile razy wykonać algorytm w pętli
Temperatura początkowa:
1000                                //poniżej definicja parametrów algorytmu
Alfa:                               //symulowanego wyżarzania
0.9
Temperatura końcowa:
1
Czas trwania epoki:
100
```

5. Procedura badawcza

Należało zbadać zależność czasu rozwiązania problemu i złożoność pamięciową od wielkości instancji dla metody symulowanego wyżarzania. W przypadku tego algorytmu w przestrzeni rozwiązań dopuszczalnych występowały parametry programu, które mogły mieć wpływ na czas i jakość uzyskanego wyniku. W związku z tym procedura badawcza polegała na uruchomieniu programu sterowanego plikiem inicjującym *.INI*, wraz z podaniem żądanych parametrów metody (jego struktura została opisana powyżej).

Dla problemów o mniejszych rozmiarach niż dziewięćdziesiąt sześć można było wykonać algorytm sto razy na jedno uruchomienie programu, celem wyciągnięcia średniego wyniku. Dla większych instancji zmniejszono tę ilość do dziesięciu, aby otrzymać wyniki w rozsądnym czasie. Pomiary wykonano dla różnych wartości parametrów algorytmu celem oszacowania złożoności dla różnych wartości błędu (100%, 70%, 50%, 30%) oraz porównania dokładności wraz z zestawieniem z innymi metodami (programowanie dynamiczne i przegląd zupełny). Oprócz tego mierzono wpływ parametrów symulowanego wyżarzania (temperatura początkowa, alfa, temperatura końcowa oraz liczba epok) na błąd oraz czas wykonywania algorytmu (na jednej instancji – *gr96.tsp*).

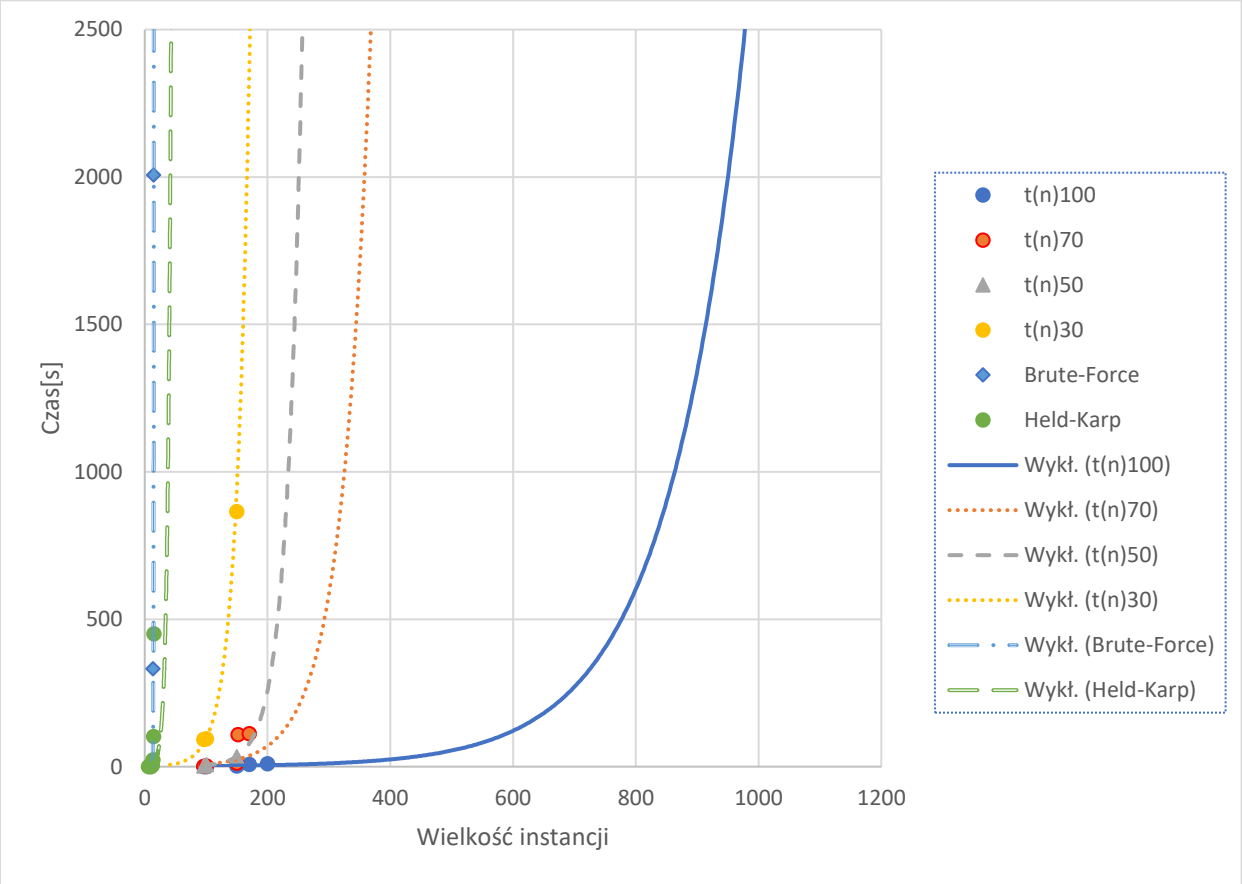
Ponadto, mierzono średnie zużycie pamięci procesu dla tego algorytmu, celem porównania złożoności pamięciowej z programowaniem dynamicznym.

Wyniki zostały zgromadzone w plikach wyjściowych *.csv* znajdujących się w katalogu *output*.

Wyniki opracowane zostały w programie MS Excel.

6. Wyniki

Wyniki zgromadzone zostały w plikach *.csv* w katalogu *output*. Przedstawione zostały w postaci wykresu zależności czasu uzyskania rozwiązania problemu od wielkości instancji (wykres 1.). Kolejne wykresy (2.–9.) przedstawiają zależności wybranych parametrów w funkcji czasu wykonywania algorytmu oraz procentowego błędu. Wykres 10. natomiast prezentuje zestawienie złożoności pamięciowej algorytmu programowania dynamicznego z symulowanym wyżarzaniem.



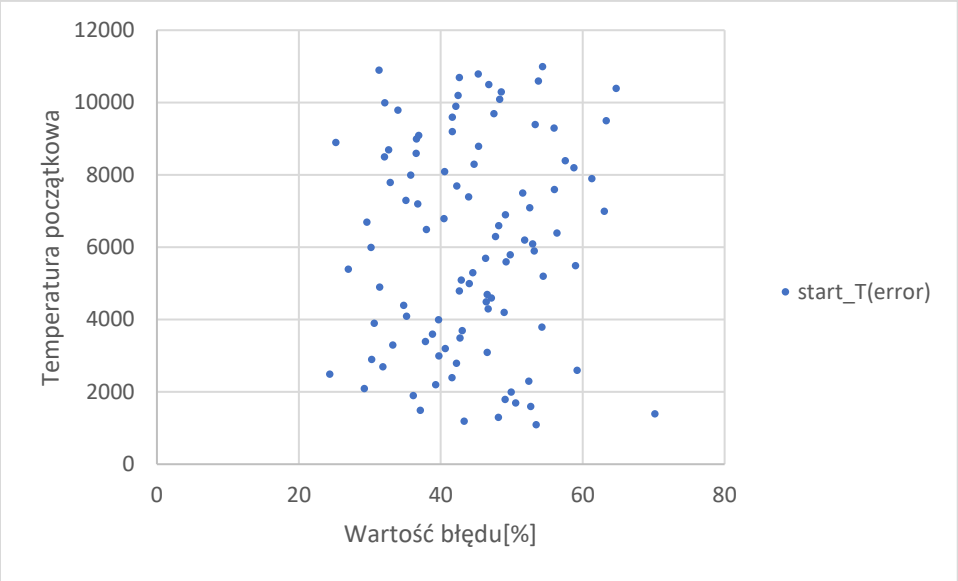
Wykres 1. Wykres wynikowy porównujący różne metody rozwiązywania TSP.

Wielkość instancji:	*Czas[s](100% błędu)*	*Czas[s](70% błędu)*	*Czas[s](50% błędu)*	*Czas[s](30% błędu)*
96	*0,2781*	*1,0412*	*2,8529*	*92,2513*
100	*0,6744*	*2,6414*	*6,6530*	*95,2932*
150	*2,4661*	*12,2728*	*33,7468*	*865,1830*
152	*6,7798*	*108,3524*	*39728,9708*	*732,0787*
170	*7,7830*	*111,6082*	*382,0348*	#
200	*9,8258*	*64,1532*	*153,5704*	*932,0787*
323	*1,8469*	*99,2363*	*103,8681*	#

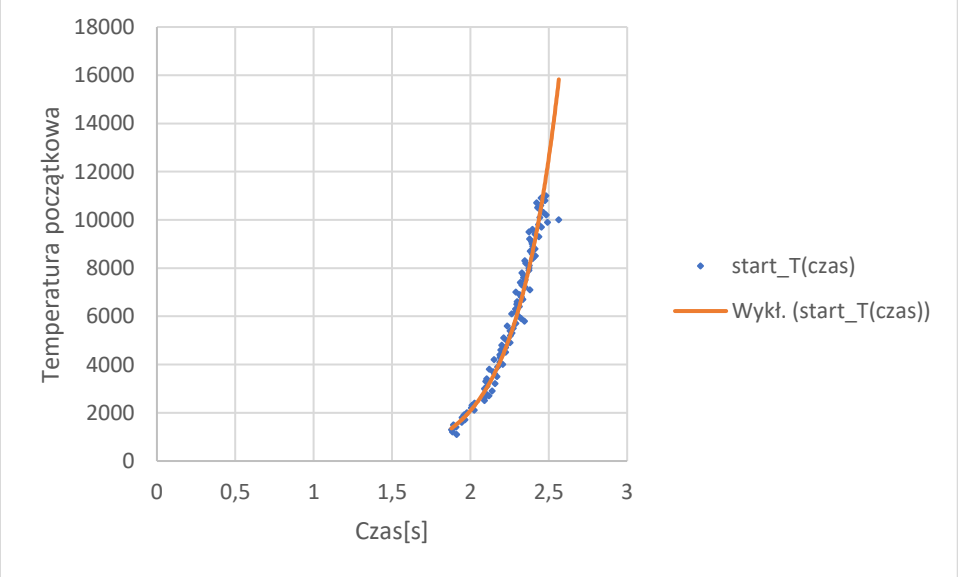
Tabela 1. Dane pomiarowe dla metody symulowanego wyżarzania.

Wykres 1. podsumowuje wszystkie metody dotychczas badane metody. Zestawia algorytm przeglądu zupełnego, programowania dynamicznego oraz symulowanego wyżarzania. Ten ostatni, ze względu na pewną losowość został przestawiony za pomocą czterech linii, każda odpowiada złożoności algorytmu dla wymuszonych parametrów w taki sposób, aby dawała błędy rzędu odpowiednio w pesymistycznym przypadku 100%, 70%, 50% oraz 30%.

Wykresy dla temperatury początkowej:

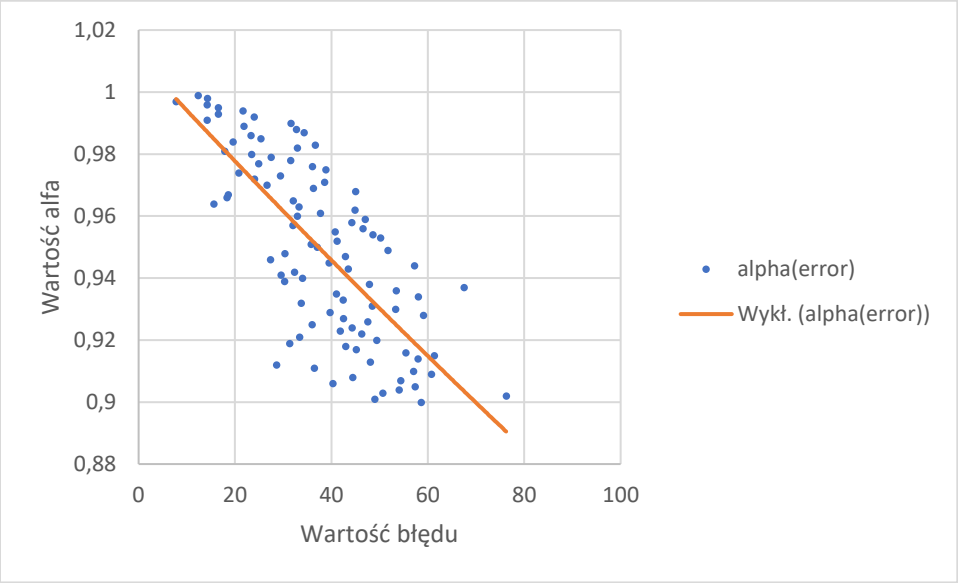


Wykres 2. Zależność temperatury początkowej od błędu procentowego.

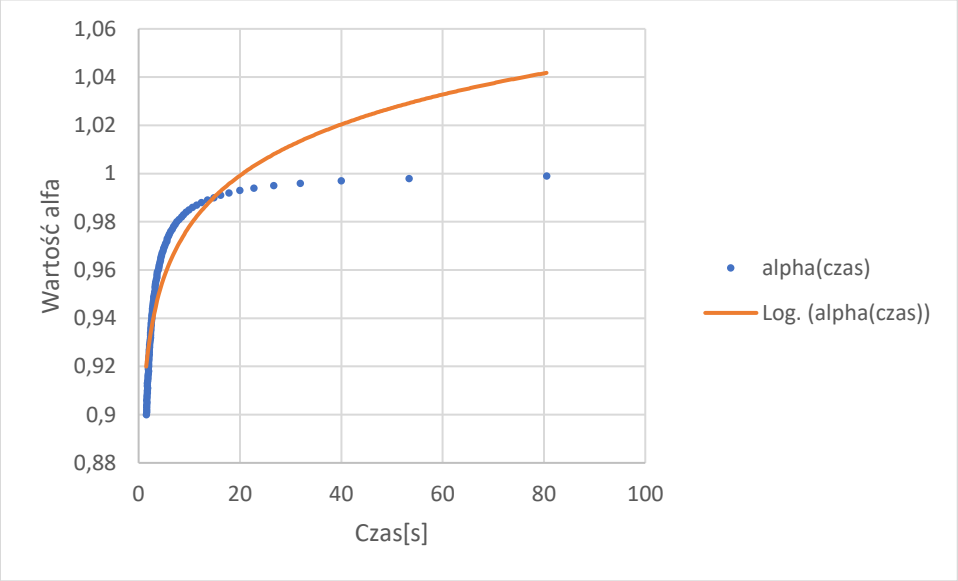


Wykres 3. Zależność temperatury początkowej od czasu wykonania.

Wykresy dla współczynnika chłodzenia:

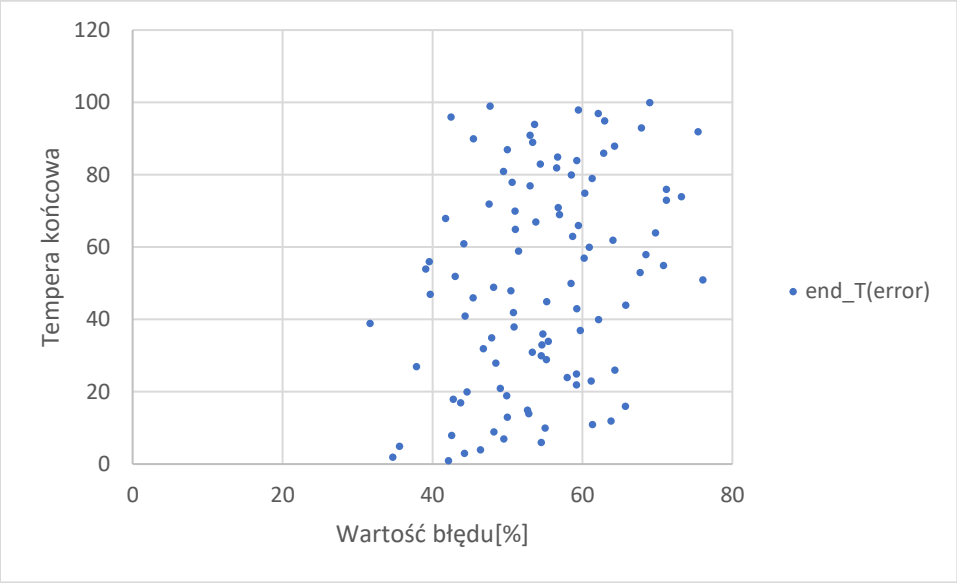


Wykres 4. Zależność współczynnika chłodzenia od wartości błędu procentowego.

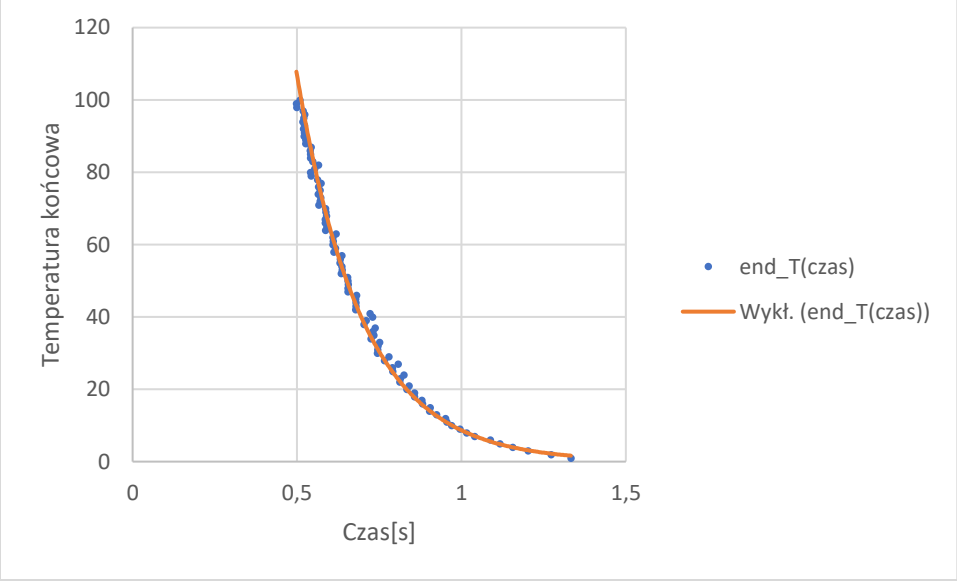


Wykres 5. Zależność współczynnika chłodzenia od czasu wykonania.

Wykresy dla temperatury końcowej:

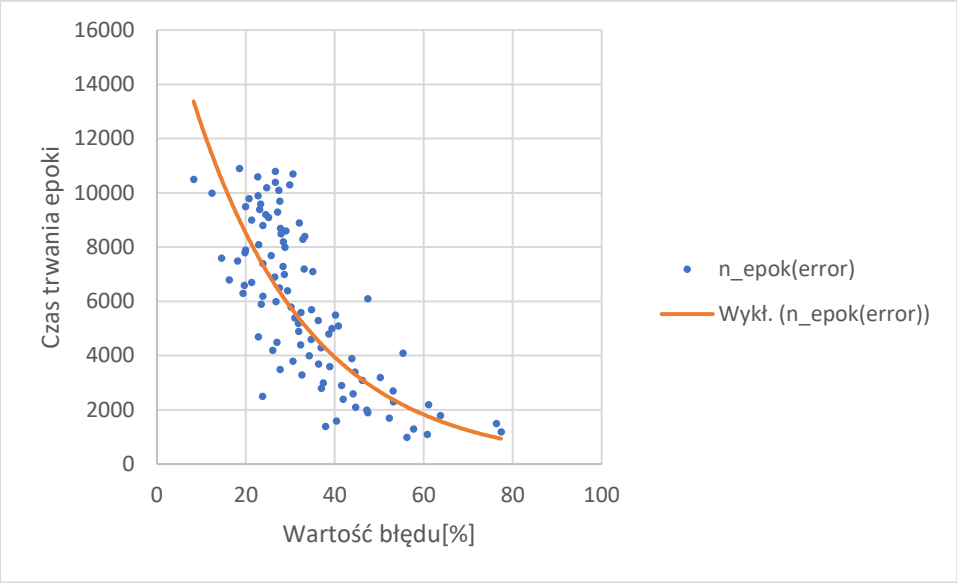


Wykres6. Zależność temperatury końcowej od błędu procentowego.

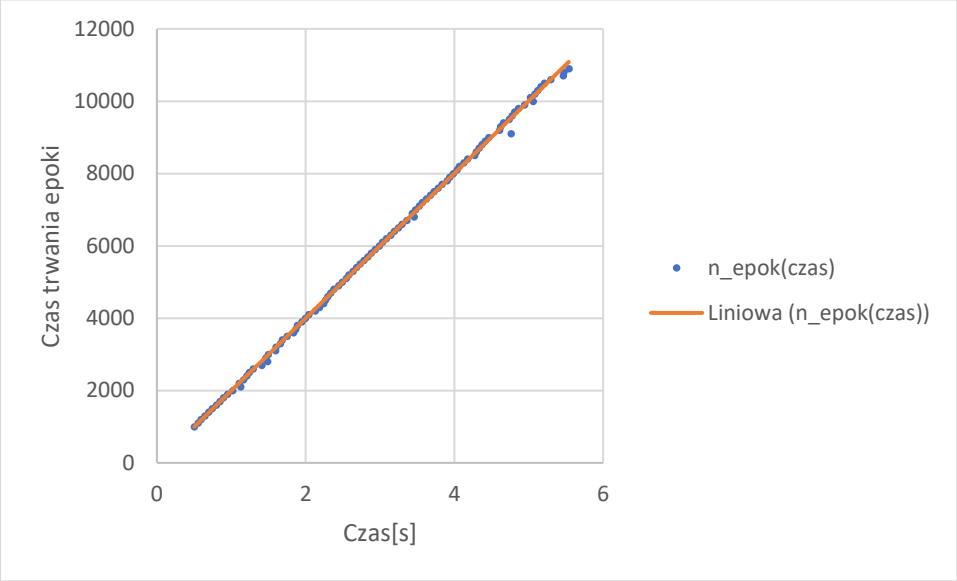


Wykres 7. Zależność temperatury końcowej od czasu wykonania.

Wykresy dla czasu trwania epoki:

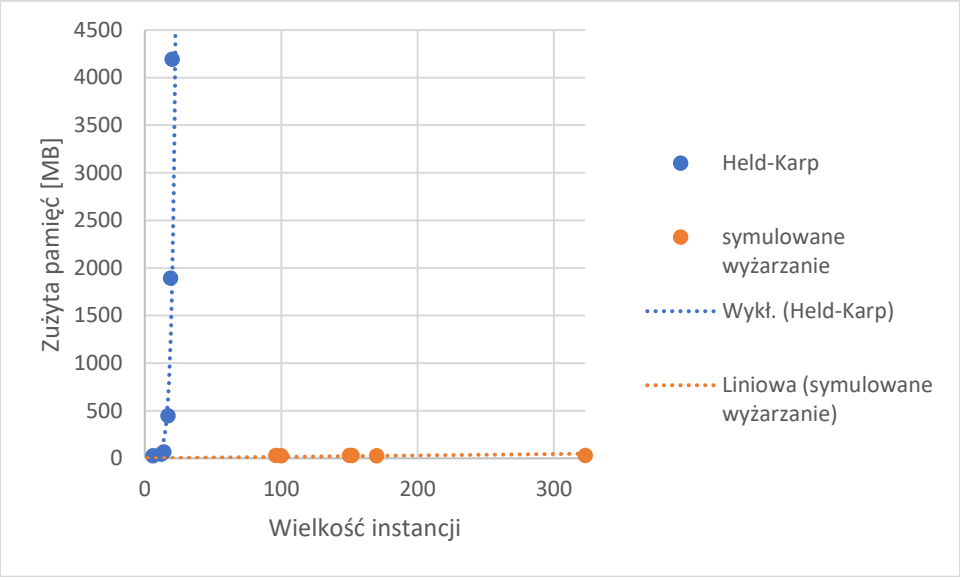


Wykres 8. Zależność czasu trwania epoki od błędu procentowego.



Wykres 9. Zależność czasu trwania epoki od czasu wykonania.

Wykres dla złożoności pamięciowej:



Wykres 10. Zestawienie złożoności pamięciowej metod symulowanego wyżarzania oraz programowania dynamicznego

Programowanie dynamiczne		*Symulowane wyżarzanie*	
Wielkość instancji	*Zużywana pamięć[MB]*	*Wielkość instancji*	*Zużywana pamięć[MB]*
6	*28*	*96*	*30*
12	*44*	*100*	*29*
14	*67*	*150*	*30*
17	*447*	*152*	*31*
19	*1896*	*170*	*29*
20	*4194*	*323*	*31*

Tabela 2. Dane pomiarowe zużycia pamięci przez metodę programowania dynamicznego oraz symulowanego wyżarzania

7. Analiza wyników i wnioski

Metoda symulowanego wyżarzania pozwala na wyliczanie instancji nawet dwadzieścia razy większych w czasie mniejszym niż instancje rzędu 20 dla metody Helda-Karpa oraz 13 dla metody przeglądu zupełnego. Pomimo faktu, że rozwiązania praktycznie zawsze podawane są z błędem, przez wzgląd na szybsze czasy wykonywania oraz możliwość wyznaczenia rozwiązania większych problemów, wyniki algorytmu dają zadowalające rezultaty.

Przez mnogość parametrów, które wpływają zarówno na czas wykonywania, jak i wielkość błędu trudno wyznaczyć dokładną złożoność tej metody, jednak patrząc na wykres 1., widać, że jest ona wykładnicza.

Po wykresie drugim widać, że temperatura początkowa wpływa na dokładność metody w trudny do przewidzenia sposób. Świadczy o tym fakt, iż zebrane dane nie wykazują żadnej konsekwencji w swoim ułożeniu. Można natomiast łatwo zauważyć, że wraz ze wzrostem temperatury początkowej czas wykonywania programu rośnie w funkcji quasi-wykładniczej (wykres 3.).

Wykres 4. wskazuje na pewną konsekwencję podczas modyfikacji współczynnika chłodzenia, który gdy maleje, daje mniejsze błędy. Nie jest to jednak pewne przez wzgląd na duży rozrzut wyników.

Na wykresie 5. Można zauważyć, że czas wykonywania programu rośnie w funkcji logarytmicznej wraz ze wzrostem alfy.

Podobnie jak wykres 2., wykres 6. pokazuje brak konsekwencji w rozłożeniu wyników dla temperatury końcowej. Wykres 7. pokazuje, że gdy temperatura końcowa maleje, skraca tym samym czas działania algorytmu.

Wykres 8. i 9. pokazują wpływy zmiany czasu trwania epoki na błąd oraz czas trwania programu. Wartości błędu zdają się rosnąć wraz z maleniem L, jednak wyniki posiadają pewną niedokładność przez co nie jest to pewne stwierdzenie.

W porównaniu z algorytmem Helda-Karpa, symulowane wyżarzanie wypada bardzo dobrze pamięciowo, wykres 10. prezentuje praktycznie stałą złożoność pamięciową, podczas, gdy programowanie dynamiczne rośnie w funkcji wykładniczej.

Czasami zdarzają się instancje, których rozwiązanie trwa krócej, pomimo znacznie większego rozmiaru (np. *rbg323.atsp*), może to mieć związek z kosztem połączeń lub ich rozstawieniem.