PROJEKTOWANIE EFEKTYWNYCH ALGORYTMÓW

PROJEKT

11/12/2021

252736 Hutnik Szymon

Simulated Annealing (4)

1. Treść zadania

Opracować, napisać, zbadać rozwiązanie problemu komiwojażera w wersji optymalizacyjnej algorytmem *Simulated Annealing*.

Problem komiwojażera (eng. *Travelling Salesman Problem*) polega na znalezieniu minimalnego cyklu Hamiltona (przejście przez wszystkie wierzchołki tylko raz, startując i kończąc w tym samym punkcie) w pełnym grafie ważonym.

2. Opis metody

Metoda symulowanego wyżarzania polega na modyfikowaniu rozwiązania, aż temperatura spadnie do momentu „zastygnięcia" wyniku – jest to odniesieniem do zjawiska stygnięcia metali. Jeżeli spadek temperatury jest wystarczający wolny to cząsteczki tworzą równomierną strukturę. W termodynamice do opisania tego zjawiska wykorzystywany jest następujący wzór:

$$P(E) \approx e^{-\frac{E}{kT}}$$

Gdzie: k jest stałą Boltzmanna.

Wzór ten, w trochę zmienionej formie, wykorzystywany jest do obliczenia prawdopodobieństwa wybrania rozwiązania gorszego. Algorytm symulowanego wyżarzania bazuje na metodzie iteracyjnej. Metoda ta zakłada, że ciągle ulepszamy istniejące rozwiązanie, aż w końcu nie będzie można go dalej poprawić. Przejście z jednego rozwiązania do drugiego polega na znalezieniu lepszego rozwiązania sąsiedniego, czyli takiego, który znajduje się w sąsiedztwie wcześniejszego. W algorytmie symulowanego wyżarzania istnieje jednak możliwość wyboru gorszego rozwiązania z pewnym prawdopodobieństwem. Szansa na wybór gorszego rozwiązania jest zależna od temperatury, która maleje wraz z czasem wykonywania programu. Taki mechanizm sprawia, że podczas wykonywania algorytmu będzie możliwość „wyjścia" z minimum lokalnego, czyli zastąpienie minima lokalnego rozwiązaniem gorszym.

W napisanej przeze mnie implementacji zdecydowałem się na chłodzenie geometryczne, które osiągało najlepsze rezultaty.

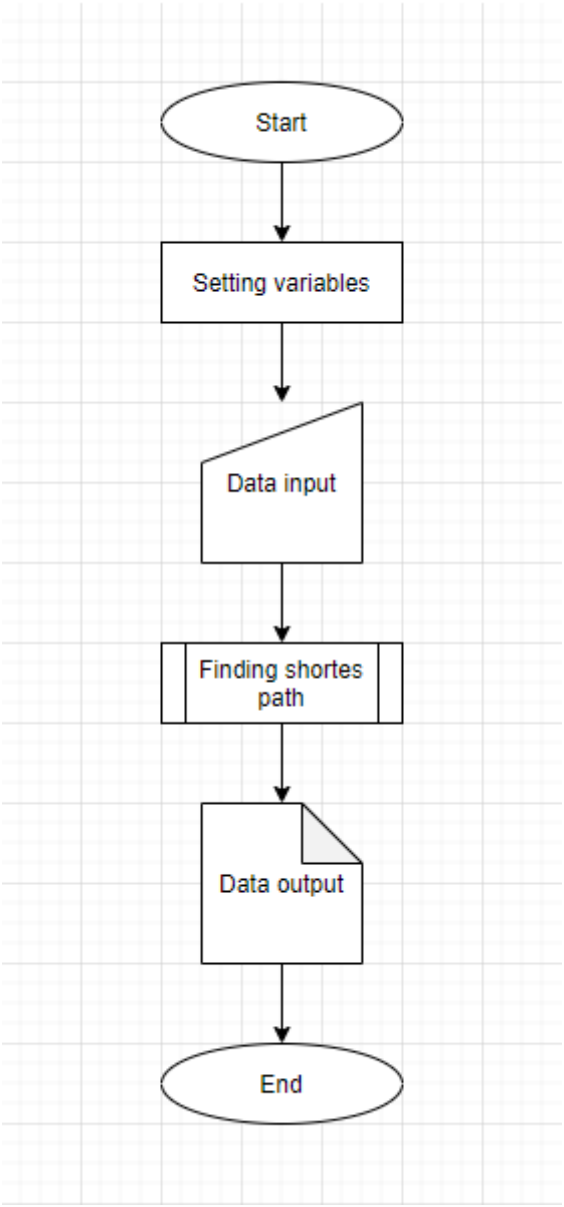
$$temperature = temperature * coolingRate^{epoch}$$

Zbadane sposoby zamiany wierzchołków to:

- zamiana 2 wierzchołków – nieefektywny dla dużych wartości, odpowiedni dla niskich
- odwrócenie trasy między 2 wierzchołkami
- zmiana odcinka między 2 wierzchołkami

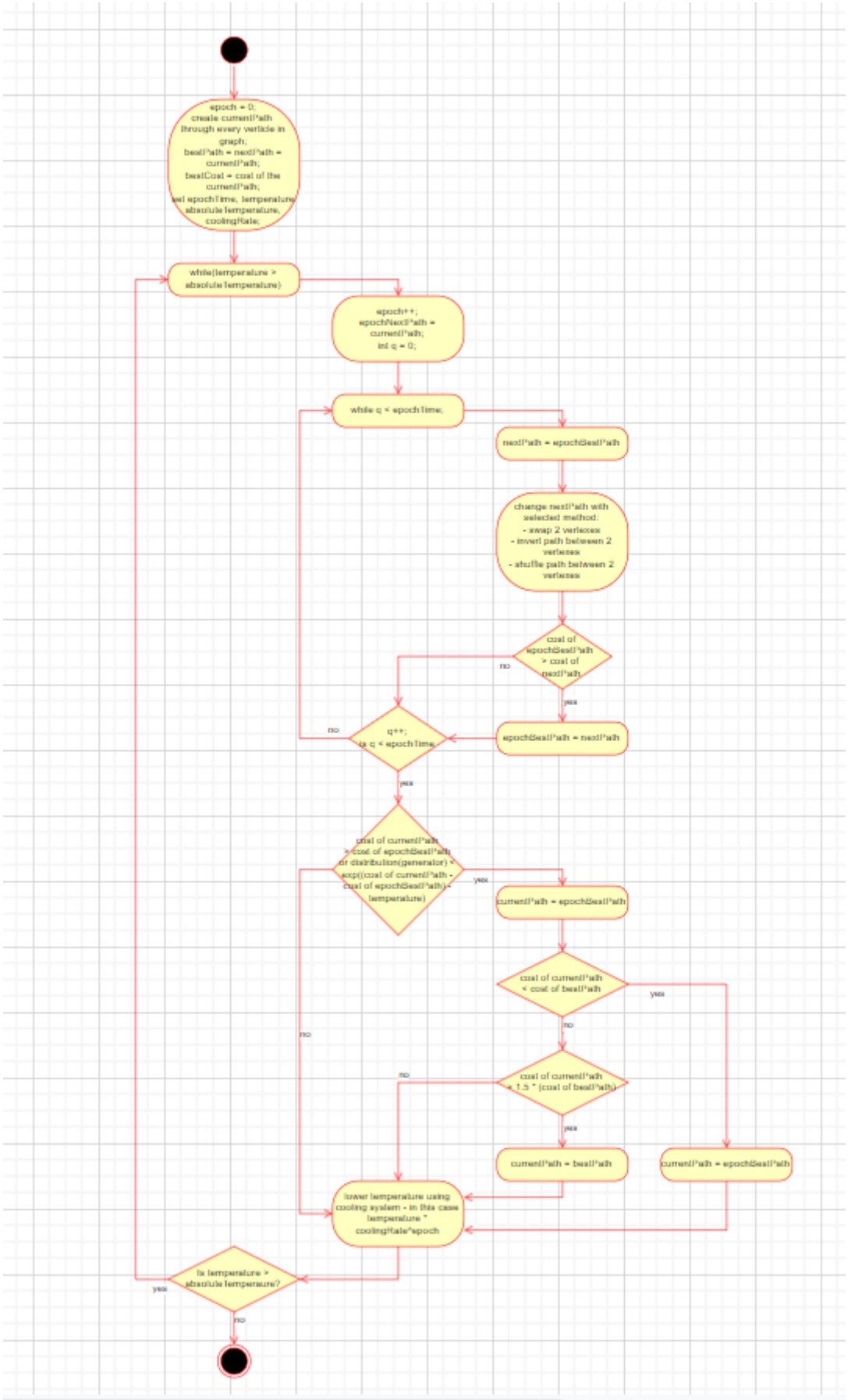
3. Opis algorytmu

Rozwiązanie zaimplementowano w postaci programu opisanego przez poniższe diagramy:



Rysunek 1: Ogólny diagram czynności programu

Najpierw inicjalizowane są zmienne. Po wczytaniu danych z konsoli następuje uruchomienie właściwej części algorytmu, następnie wypisywany jest wynik oraz czas wykonania właściwego algorytmu.



Rysunek 2: Szczegółowy diagram czynności algorytmu

4. Dane testowe

Dane, na których była badana efektywność algorytmu pochodzą ze zbioru udostępnionego przez dr Rudego. Do badania użyto wartości z następujących plików:

- m15.txt, 259
- gr21.txt, 2707
- gr48.txt, 5046
- gr96.txt, 55209
- lin105.txt, 14379
- gr137.txt, 69853
- gr202.txt, 69853
- lin318.txt, 42029
- gr431.txt, 171414

http://jaroslaw.rudy.staff.iiar.pwr.wroc.pl/files/pea/instances.zip

5. Procedura badawcza

Należało zbadać zależność czasu rozwiązania problemu od wielkości instancji. W przypadku algorytmu symulowanego wyżarzania wybrałem następujące parametry:

- $temperature = 2^n * 1000$
- $coolingRate = 0.999999999$
- $absoluteTemperature = 0.00000000001$

Każda z instancji została wykonana 50 razy z użyciem 3 różnych metod wyboru nowej trasy, udało się zbadać wszystkie wybrane do badań pliki. Wyniki były zapisywane w Excelu, następnie na ich podstawie została przeprowadzona analiza.

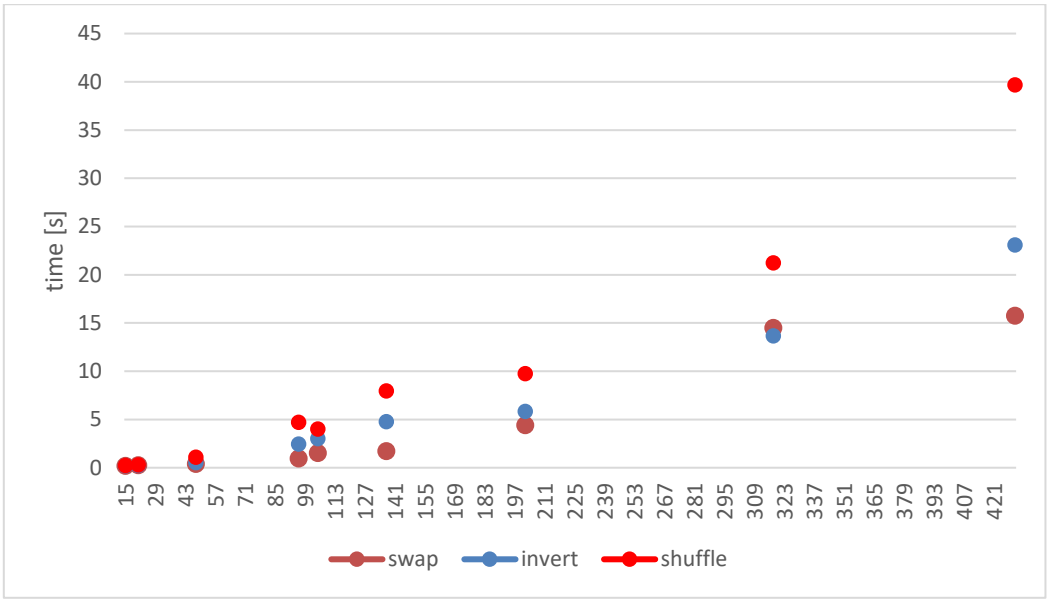
Pomiar czasu został wykonany przy użyciu bilbioteki chrono. Po otrzymaniu wyniku należy go podzielić przez liczbę powtórzeń wywołań algorytmu.

```
auto startTime = chrono::steady_clock::now();

for (int q = 0; q < repeats; ++q)
{
    cout << "Iteration " << q << endl;
    bestCost = INT_MAX;
    findShortestPath();
    totalBestCost += bestCost;
}
auto resultTime = chrono::steady_clock::now() - startTime;

double bestCostDiff = 0;
bestCostDiff = (((double)totalBestCost / (double)repeats) - (double)result) * (double)100 / (double)result;
cout << bestCost << " - expected: " << result << " difference: " << bestCostDiff << endl;
cout << chrono::duration <double>(resultTime).count()/repeats << "s \n";

while (1);
```

Item	Value
OS Name	Microsoft Windows 11 [redacted]
Version	10.0.22471 Build 22471
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	SUPERCIUPERPC
System Manufacturer	Micro-Star International Co., Ltd.
System Model	MS-7C95
System Type	x64-based PC
System SKU	To be filled by O.E.M.
Processor	AMD Ryzen 5 3600 6-Core Processor, 3600 Mhz, 6 Core(s), 12 Logical Proces...
BIOS Version/Date	American Megatrends International, LLC. 2.82, 22/06/2021
SMBIOS Version	2.8
Embedded Controller Version	255.255
BIOS Mode	UEFI
BaseBoard Manufacturer	Micro-Star International Co., Ltd.
BaseBoard Product	B550M PRO-VDH (MS-7C95)
BaseBoard Version	1.0
Platform Role	Desktop
Secure Boot State	Off
PCR7 Configuration	[redacted]
Windows Directory	
System Directory	
Boot Device	
Locale	United Kingdom
Hardware Abstraction Layer	[redacted]
Username	
Time Zone	
Installed Physical Memory (RAM)	16.0 GB
Total Physical Memory	15.9 GB
Available Physical Memory	8.21 GB
Total Virtual Memory	18.3 GB
Available Virtual Memory	6.31 GB
Page File Space	2.38 GB
Page File	[redacted]
Kernel DMA Protection	Off
Virtualisation-based security	Not enabled
Device Encryption Support	Elevation Required to View
Hyper-V - VM Monitor Mode E...	Yes
Hyper-V - Second Level Addres...	Yes
Hyper-V - Virtualisation Enable...	Yes
Hyper-V - Data Execution Prote...	Yes

6. Wyniki

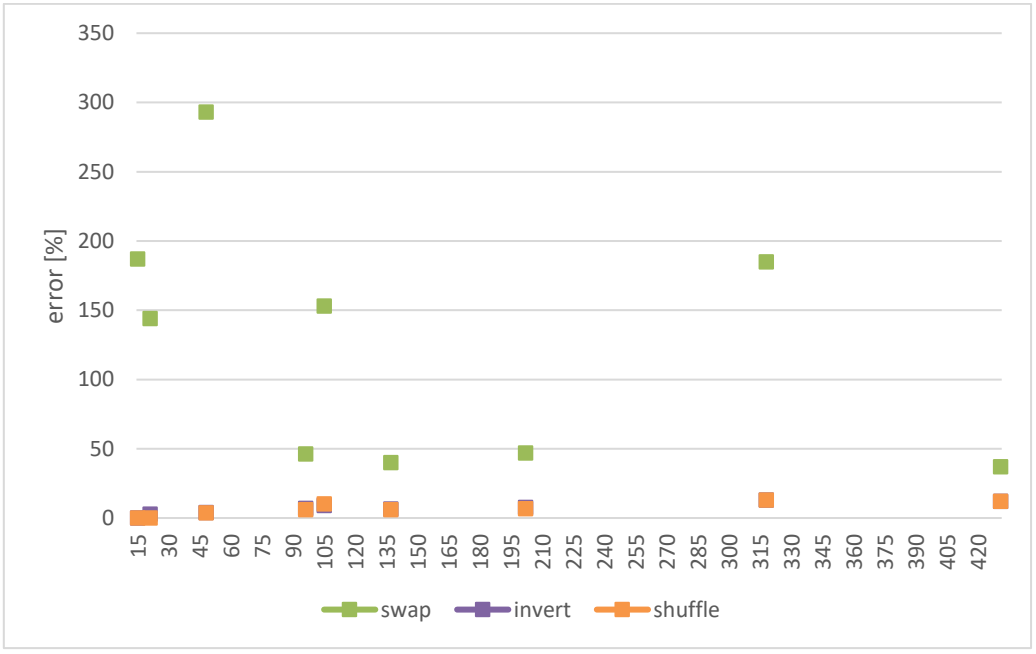
Wyniki opracowano w programie Excel:



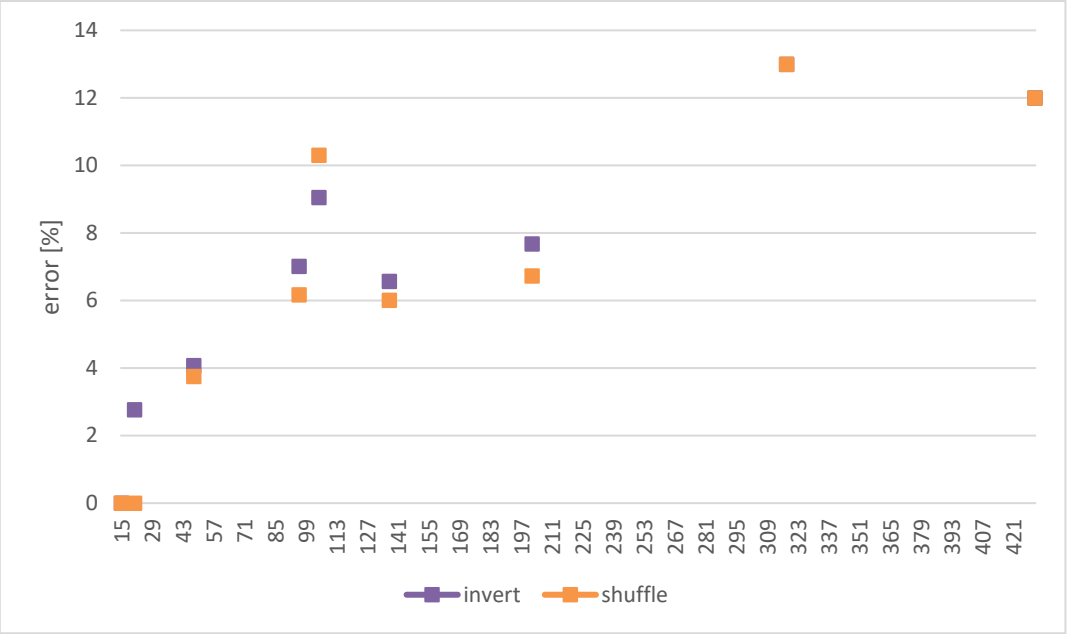
Rysunek 3: Czasy działania algorytmu dla *n* wierzchołków w zależności od metody- graf

		15	21	48	96	105	137	202	318	431
swap	time [s]	0.1949	0.2289	0.3784	0.9344	1.5148	1.707	4.4083	14.487	15.739
	error [%]	187	144	293	46.3	153	40	47	185	37
invert	time [s]	0.1692	0.2064	0.5206	2.4384	3.0134	4.7437	5.8047	13.657	23.065
	error [%]	0	2.77	4.08	7.01	9.05	6.57	7.68	13	12
shuffle	time [s]	0.1956	0.2759	1.0629	4.6956	4.0005	7.9309	9.7275	21.233	39.665
	error [%]	0	0	3.75	6.17	10.3	6.01	6.73	13	12

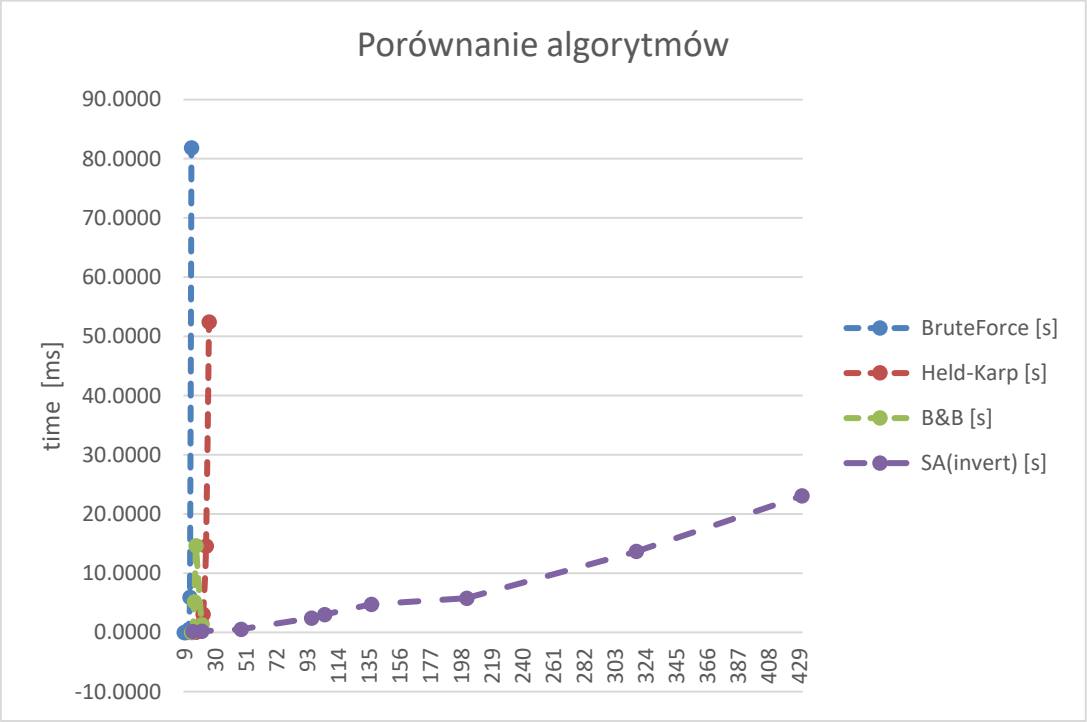
Rysunek 4: Czasy działania i błąd procentowy algorytmu dla *n* wierzchołków - tabela



Rysunek 5: Błąd procentowy dla *n* wierzchołków w zależności od metody – graf



Rysunek 6: Błąd procentowy dla *n* wierzchołków w zależności od metody (przybliżenie dla metody invert i shuffle) – graf



Rysunek 7: Porównanie czasu działania dotychczas opracowanych algorytmów dla *n* wierzchołków - graf

*dane testowe nie są kolejnymi wartościami n stąd połączenie znanych wyników linią

7. Analiza

Na podstawie przeprowadzonego badania można zauważyć, że metody invert oraz shuffle dają zdecydowanie lepsze rezultaty niż swap, w dalszej analizie będę się odwoływał do tych 2 metod. Krzywa wzrostu czasu względem wielkości instancji rośnie w przybliżeniu liniowo (Rysunek 3). Warto zwrócić uwagę, że złożoność pamięciowa w symulowanym wyżarzaniu jest pomijalna, zaś złożoność czasowa jest stosunkowo niewielka i można ją łatwo zmieniać ustawiając parametry algorytmu oraz wybierając inny algorytm chłodzenia oraz warunek stopu. SA należy do grupy algorytmów podających najlepszy osiągnięty wynik w zadanym czasie, zazwyczaj otrzymuje się błąd, który w przeprowadzonych przeze mnie badaniach kształtował się na poziomie 10% i powoli rósł dla większych instancji.

Porównanie dotychczas zrealizowanych algorytmów pokazuje jak dużą przewagę czasową ma szukanie przybliżonego rozwiązania zamiast dokładnego. Czas wykonania symulowanego wyżarzania jest bezkonkurencyjny w porównaniu z algorytmem Helda-Karpa. Takie niedokładne podejście pozwala na szukanie rozwiązań dla algorytmów o wysokiej złożoności, mimo błędu dostajemy przynajmniej przybliżony wynik.