Implementacja i analiza efektywności algorytmu Symulowanego Wyżarzania.

Zadanie projektowe nr 2

Prowadzący: dr inż. Jarosław Mierzwa

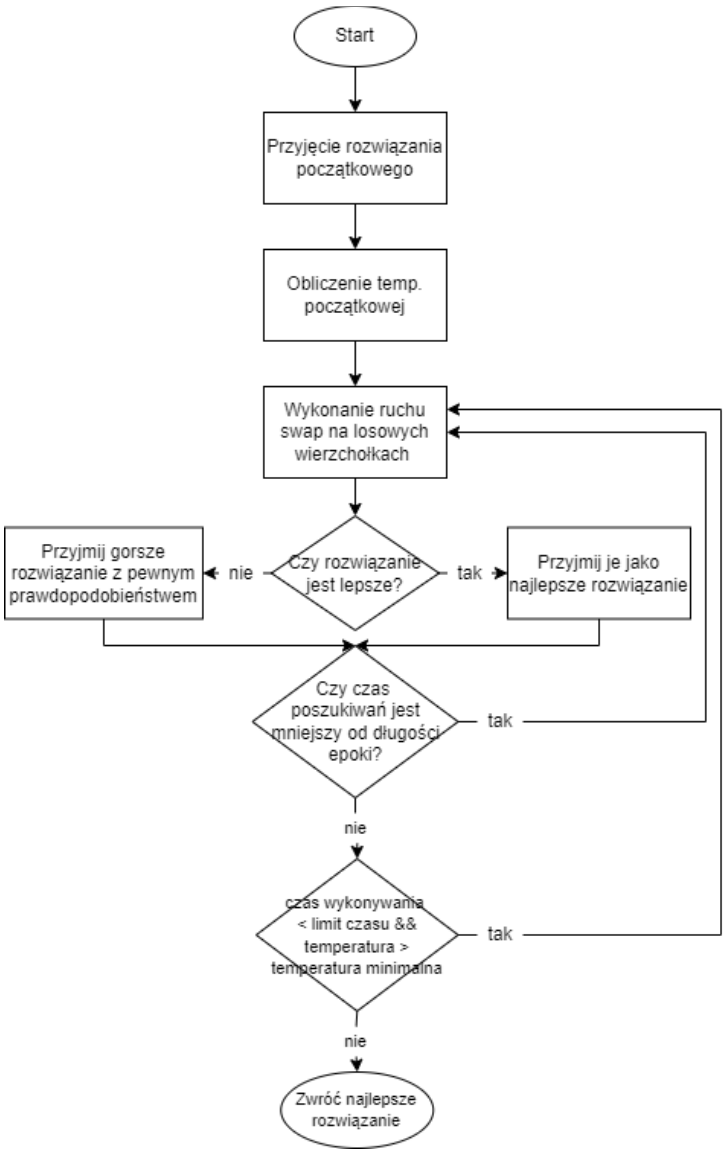
Grupa: Piątek 15.15-16.55

Autor: Izabela Pałubicka 252771

1. Wstęp teoretyczny.

1.1. Symulowane Wyżarzanie

Algorytm symulowanego wyżarzania polega na badaniu różnych obszarów przestrzeni rozwiązań. Metoda ta zakłada, że ciągle ulepszamy istniejące rozwiązanie, aż w końcu nie będzie można go dalej poprawić. W algorytmie tym istnieje jednak możliwość wyboru gorszego rozwiązania z pewnym prawdopodobieństwem. Dzięki temu algorytm w określonych warunkach może wyjść ze znalezionego minimum lokalnego i dalej podążać w kierunku rozwiązania optymalnego. Szansa na wybór gorszego rozwiązania jest zależna od temperatury, która maleje wraz z czasem wykonywania programu (chłodzenie). Im temperatura jest wyższa, tym prawdopodobieństwo wyboru gorszego rozwiązania jest większe (większa dywersyfikacja). Im niższa, tym algorytm jest bardziej zbliżony w działaniu do typowych metod iteracyjnych (większa intensyfikacja).



Rysunek 1: Schemat blokowy algorytmu Symulowanego Wyżarzania.

1.2. Rozwiązanie początkowe

Rozwiązanie początkowe jest wybierane w sposób pseudolosowy. Na początku wektor wypełniany jest wierzchołkami kolejno od 0 do N-1 włącznie kończąc na 0, a następnie wywoływana jest na nim metoda random_shuffle, pomijając tasowanie wierzchołka startowego. Następnie rozwiązanie to przypisywane jest jako najlepsze i sprawdzane jest czy w jego sąsiedztwie znajduje się lepsze rozwiązanie, jeśli tak to najlepsze z nich jest zapamiętywane i sprawdzane jest jego sąsiedztwo. Jeśli w sąsiedztwie bieżącego najlepszego rozwiązania nie znaleziono lepszego rozwiązania to funkcja kończy swe działanie i zwraca najlepszy znaleziony wynik.

```
vector<int> SW::generateInitialSolution(DataMatrix data)
{
    vector<int> currentSolution;
    vector<int> bestSolution;
    currentSolution.resize(data.size + 1);
    bestSolution.resize(data.size + 1);
    bool locallyOpitmal = false;

    // Przyjęcie rozwiązania początkowego
    for (int i = 0; i < data.size; i++) currentSolution[i] = i;
    currentSolution[data.size] = 0;
    // Przetasowanie rozwiązania
    random_shuffle(currentSolution.begin() + 1, currentSolution.end() - 1);
    int bestCost = calculatePathCost(currentSolution, data.matrix);
    bestSolution = currentSolution;
    do{
        locallyOpitmal = false;
        currentSolution = bestSolution;
        // Sprawdzanie sąsiedztwa bieżącego rozwiązania
        for (int i = 1; i < data.size -1; i++) {
            for (int j = i + 1; j < data.size; j++) {
                swap(currentSolution[i], currentSolution[j]);
                int currentCost = calculatePathCost(currentSolution, data.matrix);
                // Jeżeli rozwiązanie sąsiadujące ma lepsze rozwiązanie to zamiana
                if (currentCost < bestCost) {
                    locallyOpitmal = true;
                    bestCost = currentCost;
                    bestSolution = currentSolution;
                }
                swap(currentSolution[i], currentSolution[j]);
            }
        }
    } while (locallyOpitmal);
    return bestSolution;
}
```

Rysunek 2: Funkcja generująca rozwiązanie początkowe.

1.3. Temperatura początkowa

Gdy algorytm rozpoczyna swoje działanie jego temperatura jest wysoka, co daje mu możliwość częstszej zmiany konfiguracji rozwiązania, a prawdopodobieństwo wyboru gorszego rozwiązania jest wyższe. Sposób wyboru temperatury początkowej został przedstawiony na Rysunku 2. Za temperaturę początkową wybierana jest maksymalna różnica kosztu pomiędzy dwoma sąsiedztwami wybieranymi w sposób pseudolosowy.

```
int SW::calculateInitialTemperature(DataMatrix data, vector<int> bestPermutation, int initialTempPrecision)
{
    auto tempStart = bestPermutation;

    int initialTemp = INT32_MIN;
    for (int k = 0; k < data.size * initialTempPrecision; k++) {

        int cost1 = calculatePathCost(tempStart, data.matrix);

        int i = (rand() % (data.size - 1)) + 1;
        int j = (rand() % (data.size - 1)) + 1;
        if (i == j) continue;

        int tmp = tempStart[j];
        tempStart[j] = tempStart[i];
        tempStart[i] = tmp;

        int cost2 = calculatePathCost(tempStart, data.matrix);

        int diff = abs(cost1 - cost2);

        // Ustalenie temperatury początkowej jako maksymalna różnica pomiędzy dowolnymi sąsiadami
        if (diff > initialTemp) initialTemp = diff;

        random_shuffle(tempStart.begin() + 1, tempStart.end() - 1);
    }
    return initialTemp;
}
```

Rysunek 3: Funkcja generująca temperaturę początkową.

1.4. Schładzanie

Wraz z kolejnymi iteracjami algorytmu temperatura spada, co powoduje, że prawdopodobieństwo wyboru gorszego rozwiązania maleje, a na sam koniec działania algorytmu jest bliskie zeru.

W projekcie użyty został schemat schładzania dla redukcji geometrycznej, której wzór wynosił $T = a*T$, gdzie $a < 1$. W danej temperaturze testowanych jest wiele rozwiązań. Wartości a, w praktycznych zastosowaniach, nie powinny być mniejsze niż 0,85. W projekcie zastosowano trzy wartości współczynnika a i wynosiły one 0.85, 0.99 i 0.9999.

1.5. Definicja sąsiedztwa

Sąsiedztwo jakiegoś rozwiązania x jest zbiorem rozwiązań, które mogą być osiągnięte z drogą x przez jakąś operację. Taką operację nazywa się ruchem. Jeśli nowe rozwiązanie y jest lepsze od dowolnego z jego sąsiedztwa, wtedy y jest optimum lokalnym.

1.6. Epoka

Epoka jest ilość iteracji losowania sąsiednich wyników dla badanej temperatury. Wynosi ona: E=N*b, gdzie N to wielkość instancji, a b to podana stała, która wynosiła 300.

1.7. Warunek zakończenia

Warunkiem zakończenia algorytmu jest osiągnięcie temperatury minimalnej, która ustawiona została jako wartość 10^{-15} lub limitu czasu, który dla różnych problemów posiadał różne wartości.

- ftv47.atsp wynosił 2 min
- ftv170.astp wynosił 4 min
- rbg403.atsp wynosił 6 min.

2. Opis najważniejszych klas w projekcie

Klasa DataMatrix

Jest to klasa przechowująca dane wczytane z pliku, a także ustawiane parametry algorytmu.

```
int size; //ilość wierzchołków
int** matrix; //tabela kosztów
float coolingTemp; //współczynnik schładzania
int stopTime; //kryterium stopu
```

Rysunek 4: Wczytywane i podawane parametry programu.

Posiada również metodę wczytującą dane z pliku.

Klasa SW

Klasa ta posiada metodę runAlgorithm(), która wykonuje algorytm Symulowanego Wyżarzania. Wywołuje ona na początku swojego działania metody generateInitialSolution(), która zwraca rozwiązanie początkowe i calculateInitialTemperature() zwracającą temperaturę początkową. Metody te zostały opisane powyżej. W dalszej części metody sprawdzane są kolejne rozwiązania. Ten fragment kodu został umieszczony na Rysunku 5. Początkowo przyjmuje się najlepsze rozwiązanie jako aktualne, następnie losuje się wierzchołki, które maja zamienić się miejscami. Obliczany jest koszt, jeśli jest niższy to zostało znalezione lepsze rozwiązanie i teraz to jego sąsiedztwo będzie badane. Jeśli nie, obliczane jest prawdopodobieństwo, z jakim gorsze rozwiązanie ma szanse na to, że w jego sąsiedztwie może się znaleźć lepsze.

```
timer.startTimer();
do {
    timer.stopTimer();
    int counter = 0;

    while ((timer.timeCounter() < data.stopTime) && (counter < era))
    {
        // wykonanie ruchu swap na losowych wierzchołkach
        currentPermutation = bestPermutation;
        int i = (rand() % (data.size - 1)) + 1;
        int j = (rand() % (data.size - 1)) + 1;
        if (i == j) continue;

        int tmp = currentPermutation[j];
        currentPermutation[j] = currentPermutation[i];
        currentPermutation[i] = tmp;

        int cost = calculatePathCost(currentPermutation, data.matrix);

        // Przyjęcie rozwiązania jako najlepsze
        if (cost < bestCost)
        {
            timer.stopTimer();
            bestCost = cost;
            bestPermutation = currentPermutation;
            time = timer.timeCounter();
        }
        // Przyjęcie gorszego rozwiązania z pewnym prawdopodobieństwem
        else if (random() < exp(-(static_cast<float>((cost - bestCost) / currentTemp))))
        {
            timer.stopTimer();
            bestCost = cost;
            bestPermutation = currentPermutation;
            time = timer.timeCounter();
        }
        counter++;
    }
    currentTemp = currentTemp * data.coolingTemp;
    timer.stopTimer();
    // Warunek kończący algorytm
    // czas wykonywania < limit czasu && temperatura > temperatura minimalna
} while (timer.timeCounter() < data.stopTime && currentTemp > Tmin);
Path bestSolution = Path(bestPermutation, bestCost, time);
return bestSolution;
```

Rysunek 5: Sprawdzanie sąsiednich rozwiązań, pomiar czasu, sprawdzanie kryteriów stopu.

Klasa Timer

Jest odpowiedzialna za pomiar czasu działania całego algorytmu, a także za pomiar czasu wewnątrz funkcji dla sprawdzenia kryterium stopu, dzięki czemu algorytm może zostać przerwany. Korzysta z biblioteki chrono, a mierzony czas podawany jest w sekundach. Do pomiaru upływu czasu wykorzystana została metoda startTimer(), stopTimer(), a także timeCounter(), która zwraca różnicę czasu między początkiem a końcem odmierzania czasu.

3. Wyniki

3.1. Pomiary błędu względnego w funkcji czasu i najlepsze rozwiązanie

Błąd względny został policzony ze wzoru:

$$\delta = \frac{|f_{zn} - f_{opt}|}{f_{opt}}$$

gdzie f_{zn} – wartość policzona przez algorytm, f_{opt} – najlepsze znane rozwiązanie

Wyniki dla pliku ftv47.atsp (poprawny wynik 1776):

współczynnik schładzania	czas algorytmu[s]	błąd względny[%]	koszt	czas znalezienia rozwiązania[s]
0,85	31,13	22,97	2184	29,61
	33,88	16,10	2062	32,78
	33,43	9,12	1938	31,98
	31,30	11,94	1988	30,71
	61,59	22,92	2183	59,82
	35,83	19,71	2126	33,21
	37,90	17,46	2086	36,12
	33,16	15,54	2052	31,14
	40,29	20,61	2142	38,89
	50,43	28,49	2282	48,11
0,99	43,42	17,74	2091	41,98
	54,83	13,46	2015	53,21
	51,05	14,02	2025	49,14
	46,38	19,93	2130	45,23
	43,47	9,68	1948	41,32
	47,20	15,77	2056	45,91
	42,00	23,20	2188	39,96
	46,49	28,60	2284	44,97
	44,15	15,15	2045	42,14
	43,02	10,75	1967	41,25
0,9999	120,67	16,27	2065	119,31
	110,34	19,26	2118	108,32
	120,45	18,69	2108	119,47
	120,48	8,11	1920	118,42
	114,06	24,38	2209	113,15
	109,29	20,50	2140	108,41
	117,70	15,88	2058	116,21
	106,92	14,25	2029	104,99
	120,80	15,48	2051	118,84
	119,87	10,25	1958	117,99

Tabela 1: Pomiary błędu względnego i najlepsze wyniki wraz z ich momentem dla pliku ftv47.atsp

Wyniki dla pliku ftv170.atsp (poprawny wynik 2755):

współczynnik schładzania	czas algorytmu[s]	błąd względny[%]	koszt	czas znalezienia rozwiązania[s]
0,85	241,80	176,95	7630	237,13
	241,75	432,23	14663	237,37
	241,27	580,65	18752	237,17
	241,86	235,35	9239	237,21
	240,23	56,37	4308	237,01
	241,54	584,57	18860	238,12
	241,99	44,17	3972	238,18
	241,65	68,75	4649	238,41
	239,22	49,00	4105	236,89
	241,28	73,39	4777	238,34
0,99	272,38	689,80	21759	240,02
	269,05	714,30	22434	239,26
	271,57	757,53	23625	239,79
	274,48	766,46	23871	239,82
	270,97	626,24	20008	239,89
	266,84	477,06	15898	238,75
	271,96	493,90	16362	239,58
	274,52	58,51	4367	239,96
	272,97	51,62	4177	239,75
	275,16	57,97	4352	240,01
0,9999	281,97	834,16	25736	240,01
	284,05	800,84	24818	240,02
	287,36	914,74	27956	240,01
	294,76	832,27	25684	240,02
	284,18	876,33	26898	239,89
	281,36	851,36	26210	239,84
	280,30	861,20	26481	239,75
	261,20	816,12	25239	239,13
	253,60	887,77	27213	239,25
	260,36	817,64	25281	239,67

Tabela 2: Pomiary błędu względnego i najlepsze wyniki wraz z ich momentem dla pliku ftv170.atsp

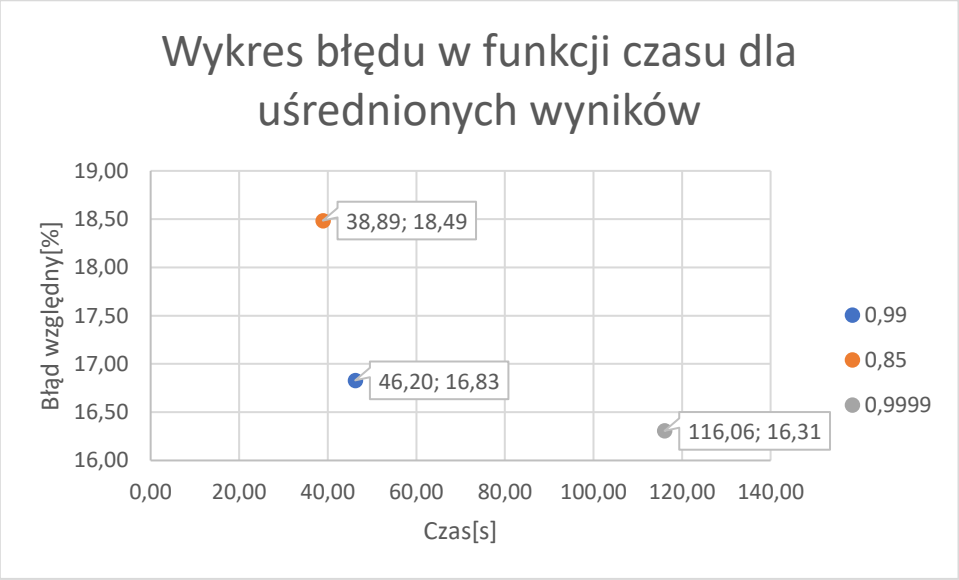
Wyniki dla pliku rbg403.atsp (poprawny wynik 2465):

współczynnik schładzania	czas algorytmu[s]	błąd względny[%]	koszt	czas znalezienia rozwiązania[s]
0,85	363,63	10,953	2735	359,12
	369,18	136,917	5840	359,68
	361,95	12,211	2766	359,14
	367,33	150,507	6175	359,56
	370,73	157,769	6354	359,78
	366,54	9,493	2699	359,85
	367,06	115,213	5305	359,87
	366,87	165,274	6539	359,95
	357,84	12,414	2771	355,12
	369,07	102,110	4982	359,56

Tabela 2: Pomiary błędu względnego i najlepsze wyniki wraz z ich momentem dla pliku rbg403.atsp

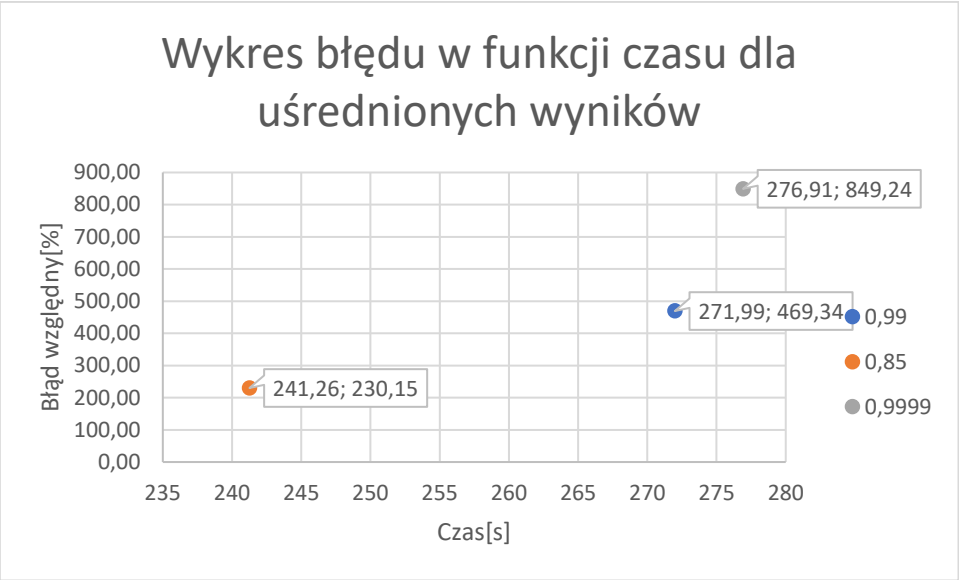
3.2. Wykres błędu w funkcji czasu dla uśrednionych wyników

Wykres dla pliku ftv47.atsp:



Wykres 1: Wykres błędu w funkcji czasu dla uśrednionych wyników dla pliku ftv47.atsp i dla współczynnika wszystkich trzech współczynników.

Wykresy dla pliku ftv170.atsp:



Wykres 2: Wykres błędu w funkcji czasu dla uśrednionych wyników dla pliku ftv170.atsp i dla wszystkich trzech współczynników.

Uśrednione dane dla pliku rbg403.atsp i współczynnika chłodzenia równego 0.85:

Uśredniony czas algorytmu[s]	Uśredniony błąd względny[%]
366,02	87,286

Tabela 4: Błąd względny w funkcji czasu dla uśrednionych wyników dla pliku rbg403.atsp i dla współczynnika 0,85.

3.3.Ścieżki dla najlepszych rozwiązań

Najlepsze rozwiązanie dla pliku ftw47.atsp:

Koszt: 1920

Droga: 0->25->10->8->11->37->38->20->18->17->13->34->35->45->19->44->15->16->46->36->14->23->12->32->7->31->30->5->6->24->4->29->3->9->33->27->28->2->41->43->42->22->39->21->40->47->26->1->0

Najlepsze rozwiązanie dla pliku ftw170.atsp:

Koszt: 3972

Droga: 0->38->35->157->33->32->158->36->31->30->28->27->26->24->25->149->148->147->137->128->164->127->126->125->130->135->151->160->150->161->142->113->115->116->117->118->119->146->145->144->143->152->14->15->159->16->21->75->11->18->19->20->37->39->156->40-

>44->45->54->58->57->63->64->56->62->61->68->167->70->87->86->108->93->166->107->165->163->99->100->102->162->103->104->114->110->109->83->69->67->85->84->71->66->65->88->153->154->89->90->91->96->98->95->92->94->106->105->97->101->123->122->120->121->124->129->136->138->139->140->141->6->7->8->9->10->76->74->12->13->17->22->23->29->34->46->47->48->59->60->50->49->2->3->4->5->133->134->131->132->111->112->169->1->81->80->79->82->78->72->73->170->51->52->53->43->55->42->41->155->168->77->0

Najlepsze rozwiązanie dla pliku rbg403.atsp:

Koszt: 2699

Droga: 0->164->25->372->278->297->111->369->196->4->151->59->246->316->120->392->228->224->360->96->338->129->351->317->50->33->376->194->380->315->345->299->49->37->125->273->357->127->396->293->335->206->267->350->336->321->289->140->77->31->6->252->399->337->32->274->46->254->141->116->91->14->69->211->147->132->243->160->3->330->290->222->385->67->105->19->18->333->268->16->47->313->7->340->142->389->217->149->51->259->130->187->262->85->163->15->78->226->379->244->80->10->103->5->64->365->169->232->261->68->209->41->230->382->212->364->303->266->159->378->377->248->186->53->73->183->17->279->95->366->292->343->158->339->189->285->223->177->250->99->197->213->356->358->370->322->107->386->235->121->391->258->134->288->215->24->54->128->255->397->301->256->86->101->367->75->284->329->294->395->240->219->236->227->320->135->270->87->42->109->178->138->57->233->324->354->200->122->401->309->108->218->349->123->192->106->165->388->348->201->118->11->238->229->225->102->56->400->157->1->207->282->281->264->112->2->61->257->375->191->162->48->161->148->210->179->182->175->300->363->362->43->9->92->146->144->137->247->36->126->283->8->216->22->21->220->145->402->287->272->332->296->193->286->74->319->88->353->39->81->342->198->174->185->221->304->394->202->119->305->276->298->291->204->167->381->302->89->195->239->117->27->371->133->265->326->154->79->62->13->387->171->63->398->176->156->390->150->34->93->84->280->12->318->170->334->245->172->26->180->359->82->249->90->72->71->253->152->76->97->188->173->115->114->312->35->361->347->263->234->65->153->310->344->331->323->168->166->199->23->393->307->143->271->237->58->277->352->231->205->110->275->325->181->346->94->311->55->184->45->66->38->306->214->241->113->83->131->355->29->251->328->139->208->373->30->124->269->308->260->374->28->136->327->104->384->383->203->60->44->190->100->98->242->70->295->155->368->52->40->341->314->20->0

4. Wnioski

Czas obliczeń zwiększa się wraz ze wzrostem temperatury chłodzenia co można zauważyć na wykresie 1 i 2. Im wyższa jest ta wartość tym algorytm dłużej się wykonuje. W przypadku ograniczenia czasu wykonywania algorytmu w postaci kryterium stopu odmierzającego czas, błąd względny może wynosić ponad 200% już w przypadku średniej wielkości instancji. Jest to spowodowane tym, że algorytm nie zdąży osiągnąć minimum lokalnego w wyznaczonym czasie. Widać to na wykresie 2, który przypomina funkcję wykładniczą. Za wysoki współczynnik chłodzenia spowodował w tym przypadku, wzrost błędu względnego. Zaś na wykresie 1 widać, że wyższy współczynnik pozwala uzyskać lepsze wyniki.

Algorytm Symulowanego Wyżarzania dla problemu komiwojażera, umożliwia znalezienie w bardzo krótkim czasie, zważywszy na duże rozmiary badanych instancji, dość satysfakcjonującego rozwiązania. Nie daje to jednak gwarancji, że rozwiązanie to będzie najlepsze.

Czynnikami, które mają wpływ na czas wykonywania programu, są: temperatura początkowa, długość ery oraz warunek zakończenia jako minimalna temperatura. Zostały ustawione przez podanie pewnej stałej i pomnożenie jej przez rozmiar instancji, które ustawiane pod konkretny problem mogłyby dać lepsze rezultaty.