Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

SPRAWOZDANIE

PROJEKTOWANIE EFEKTYWNYCH ALGORYTMÓW

Implementacja i analiza efektywności algorytmu symulowanego wyżarzania

Autor:
Klaudia Marzec #263890

Prowadzący dr inż. Dariusz Ban

1 Wstęp teoretyczny

Algorytm symulowanego wyżarzania to jedna z technik projektowania algorytmów heurystycznych (metaheurystyka), służąca do rozwiązywania problemów z dużą przestrzenią poszukiwań. Oznacza to, że nie daje on gwarancji znalezienia optymalnego rozwiązania, ale stara się znaleźć rozwiązanie zbliżone do optymalnego w akceptowalnym czasie. Cechą charakterystyczną tej metody jest występowanie parametru sterującego zwanego temperaturą, który maleje w trakcie wykonywania algorytmu. Im wyższą wartość ma ten parametr, tym bardziej chaotyczne mogą być zmiany. Podejście to jest inspirowane zjawiskami obserwowanymi w metalurgii – im większa temperatura metalu, tym bardziej jest on plastyczny.

Jest to metoda iteracyjna: najpierw losowane jest pewne rozwiązanie, a następnie jest ono w kolejnych krokach modyfikowane. Jeśli w danym kroku uzyskamy rozwiązanie lepsze, wybieramy je zawsze. Istotną cechą symulowanego wyżarzania jest jednak to, że z pewnym prawdopodobieństwem może być również zaakceptowane rozwiązanie gorsze (ma to na celu umożliwienie wyjście z maksimum lokalnego).

Prawdopodobieństwo przyjęcia gorszego rozwiązania wyrażone jest wzorem $e^{\frac{f(X)-f(X')}{T}}$ (rozkład Boltzmanna), gdzie:

- X - poprzednie rozwiązani
- X' - nowe rozwiązanie
- f - funkcja oceny jakości

Im wyższa wartość f(X), tym lepsze rozwiązanie. Ze wzoru można zauważyć, że prawdopodobieństwo przyjęcia gorszego rozwiązania spada wraz ze spadkiem temperatury i wzrostem różnicy jakości obu rozwiązań.

Złożoność obliczeniowa algorytmu symulowanego wyżarzania dla problemu komiwojażera jest trudna do jednoznacznego określenia, ponieważ zależy ona od wielu czynników, między innymi od przyjętych parametrów algorytmu (temperatura początkowa, liczba iteracji, warunek końcowy, czy parametr schładzania). Podczas implementacji algorytmu przyjęto takie parametry jak temperatura startowa, współczynnik schładzania α oraz warunek końcowy, gdy temperatura spadnie poniżej 0.1. Na tej podstawie złożoność obliczeniowa algorytmu wynosi: $O(n)$, gdzie n oznacza ilość iteracji zależącą od ustawionej tmeperatury oraz współczynnika α dla danej instancji.

2 Przykład praktyczny - Opis działania algorytmu krok po kroku

Przykład praktyczny został przedstawiony dla instancji o rozmiarze N=9. Parametry zostały tak dobrane, aby móc znaleźć najlepsze rozwiązanie.

MIASTA

	0	1	2	3	4	5	6	7	8
0	0	912	780	680	318	64	858	828	587
1	942	0	766	819	639	511	592	718	588
2	744	730	0	557	98	307	364	749	457
3	712	863	544	0	950	802	443	527	918
4	343	682	86	933	0	424	211	690	382
5	30	474	305	779	430	0	737	139	346
6	828	628	393	411	244	702	0	539	167
7	780	714	721	488	717	95	561	0	289
8	571	602	499	898	369	324	139	321	0

Rysunek 1: Macierz grafu

Przed rozpoczęciem działania algorytmu należy ustalić parametry początkowe:

- początkowa wartość temperatury T = 80

  W przypadku małych przestrzeni rozwiązań lepiej jest obniżyć temperaturę początkową i zwiększyć szybkość chłodzenia, ponieważ skróci to czas symulacji bez utraty jakości

- Współczynnik obniżania temperatury $\alpha = 0.999$, która jest obliczana za pomocą równania: $t' = t * \alpha$

- Warunek końcowy - Algorytm kończy się, gdy temperatura spadnie do 0.1

Algorytm na początku wybiera losową sekwencję miast i przyjmuje ją jako najlepsze rozwiązanie. Następnie dokonuje losowego wybrania następnej sekwencji miast - w zaimplementowanym algorytmie odbywa się to jako zamienienie miejscami losowo wybranych dwóch miast w obecnej sekwencji. Jeżeli ma ona mniejszy koszt niż aktualnie najlepsze rozwiązanie, wynik jest aktualizowany i aktualnie rozpatrywana sekwencja pozostaje taka, jaka była. Jeżeli natomiast wynik nie jest lepszy następuje obliczenie prawdopodobieństwa przyjęcia gorszego rozwiązania, poprzez podstawienie do wzoru Boltzmana:
$p = e^{\frac{f(X)-f(X')}{T}} = e^{\frac{currentDistance-bestDistance}{T}}$ To prawdopodobieństwo maleje wraz ze spadkiem temperatury, co ma na celu umożliwienie algorytmowi uniknięcia utknięcia w lokalnym minimum. Następnie losowana jest liczba z przedziału [0,1) i sprawdzany jest warunek, czy prawdopodobieństwo jest od niej mniejsze. Jeśli tak, następuje odwrócenie początkowej zamiany miast. Na koniec następuje zmniejszenie temperatury i pętla się powtarza.

3 Opis implementacji algorytmu

W celu prawidłowego zimplementowania algorytmu została użyta klasa Graf zawierająca informacje o strukturze grafu oraz klasa Macierz, która przechowuje dwuwymiarową tablicę dynamiczną krawędzi w grafie. Ponadto klasa Graf zawiera najważniejsze dla algorytmu parametry - temeraturę początkową, ilości iteracji oraz współczynnik schładzania:

```
class Graf
{

public:
    int rozmiarw = 0;
    int *wierzcholki;                   // wskaznik na poczatek tablicy wierzcholkow

    Macierz macierz;

    // PARAMETRY SIMULATED ANNEALING

    int wierzcholek;

    // poczatkowa wartosc temperatury
    double startingTemp = 80;  //80

    int iterations = 700;   //500

    // Wspolczynnik - mozliwosc wprowadzenia przez uzytkownika
    double alpha = 0.999;

    // konstruktor grafu

    Graf(int w);

    // Uzupelnienie tablic

    void completeWierzcholki();
```

Rysunek 2: Klasa Graf

Dodatkowo stworzona została klasa Travel odpowiadająca za generowanie trasy oraz manipulowanie nią. Zawiera metody odpowiadające za generowanie ścieżki, zamienianie losowych miast miejscami oraz dokonywanie odwrócenia zamiany:

```
class Travel
{
public:
    int start;
    int sizel;
    int *travel;
    int city1, city2;                   // indeksy miast do zamiany
    HANDLE hOut1 = GetStdHandle( STD_OUTPUT_HANDLE );

    Travel(int roz, int start);

    void display();

    void generateTravel();

    // Metoda zamieniajaca losowe dwa miasta w sciezce

    void swapCities();

    void revertSwap();

    int* returnTravel();

    ~Travel();
};
```

Rysunek 3: Klasa Travel

4 Plan eksperymentu

W celu sprawdzenia poprawności działania algorytmu, dokonano pomiaru czasu działania w zależności od rozmiaru problemu N. Testy efektywności algorytmu zostały przeprowadzone na losowo wygenerowanych grafach o rozmiarach N: 8, 9, 10, 11, 12, 17, 33, 38, 44, 55, 64, 70. Każdy pomiar został powtórzony 100 razy.

Generowanie grafu

Każdy graf, na którym były wykonywane pomiary, został losowo wygenerowany za pomocą odpowiedniej funkcji uzupełniającej strukturę grafu losowymi wartościami, z tym faktem, że uzupełniającą go w taki sposób, aby odległości między miastem A->B oraz B->A miały wartości zbliżone do siebie z maksymalną różnicą 50.

```
// Uzupe?nienie macierzy

for(int i=0; i<wierzcholki; i++)
{
    for(int j=0; j<wierzcholki; j++)
    {
        if(graf.matrixValue(i, j) == -1)
        {
            // Uzupelnienie

            if(i == j)
            {
                droga = 0;
                droga2 = 0;
            }
            else
            {
                droga = rand() % 1000 + 1;          // Losowanie odleglosci z zakresu <1, 1000>

                // Wylosowanie odleglosci przeciwnej

                // Losowanie znaku + lub -

                int znak = rand() % 2;

                // 0 -
                // 1 +

                // Losowanie roznicy

                int roznica = rand() % 51;

                if(znak == 0)
                    droga2 = droga - roznica;
                else
                    droga2 = droga + roznica;
            }

            graf.completeMacierz(droga, i, j);

            // Uzupelnienie odbicia lustrzanego

            graf.completeMacierz(droga2, j, i);
        }
```

Rysunek 4: Losowe generowanie grafów

Pomiar czasu

Do pomiarów czasu została wykonana operacja QueryPerformanceCounter- start przed wywołaniem danej funkcji, koniec zaraz po zakończeniu jej działania. Wyniki były zapisywane w pliku tekstowym.

5 Wyniki eksperymentu

Pierwsza tabela przedstawia przyjete wartości parametrów dla każdego rozmiaru instancji tak, aby otrzymane wyniki były możliwie jak najlepsze.

N	startingTemp	alpha
17	150	0.99
33	80 000	0.9999
38	90 000	0.99999
44	100 000	0.99999
55	1 000 000	0.999999
64	3 000 000	0.999999
70	5 000 000	0.999999

Tabela 1: Parametry

Poniższa tabela przedstawia liczbę iteracji algorytmu Symulowanego Wyżarzania dla badanych rozmiarów.

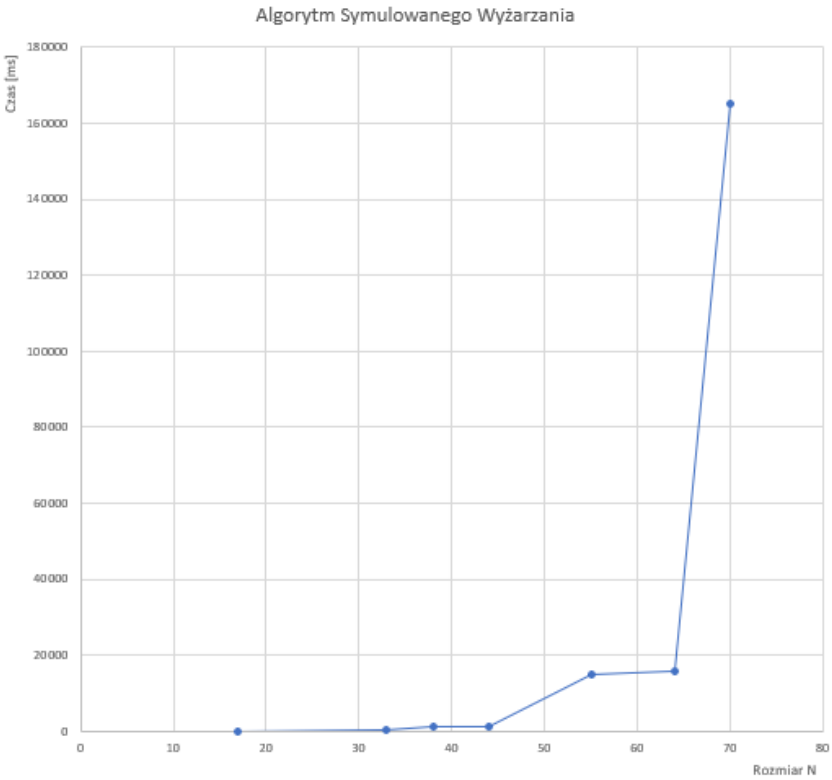
N	iteracje
17	11508
33	135917
38	1371009
44	1381545
55	16811235
64	17216700
70	152018042

Tabela 2: Liczba iteracji

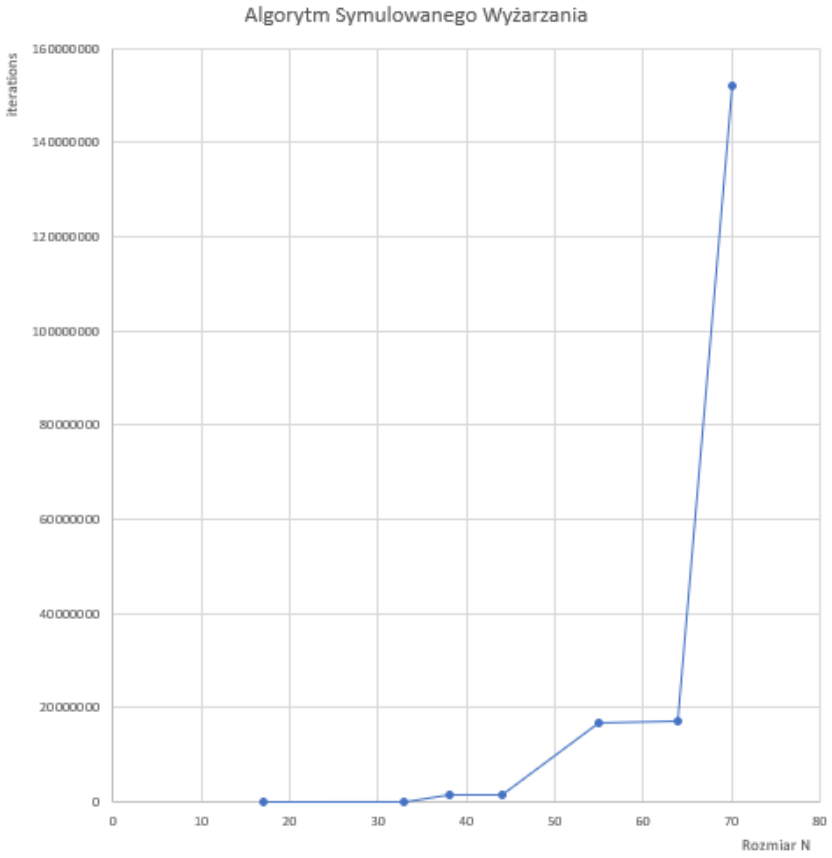
Tabela przedstawia uśrednione wyniki dla każdego badanego rozmiaru dla algorytmu Symulowanego Wyżarzania. Wyniki w tabeli zostały przedstawione w milisekundach.

N	t[ms]
17	1.122
33	77.371
38	968.681
44	991.09
55	14974.5
64	15757.5
70	165118

Tabela 3: Pomiar czasu



Rysunek 5: Symulowane Wyżarzanie - czas



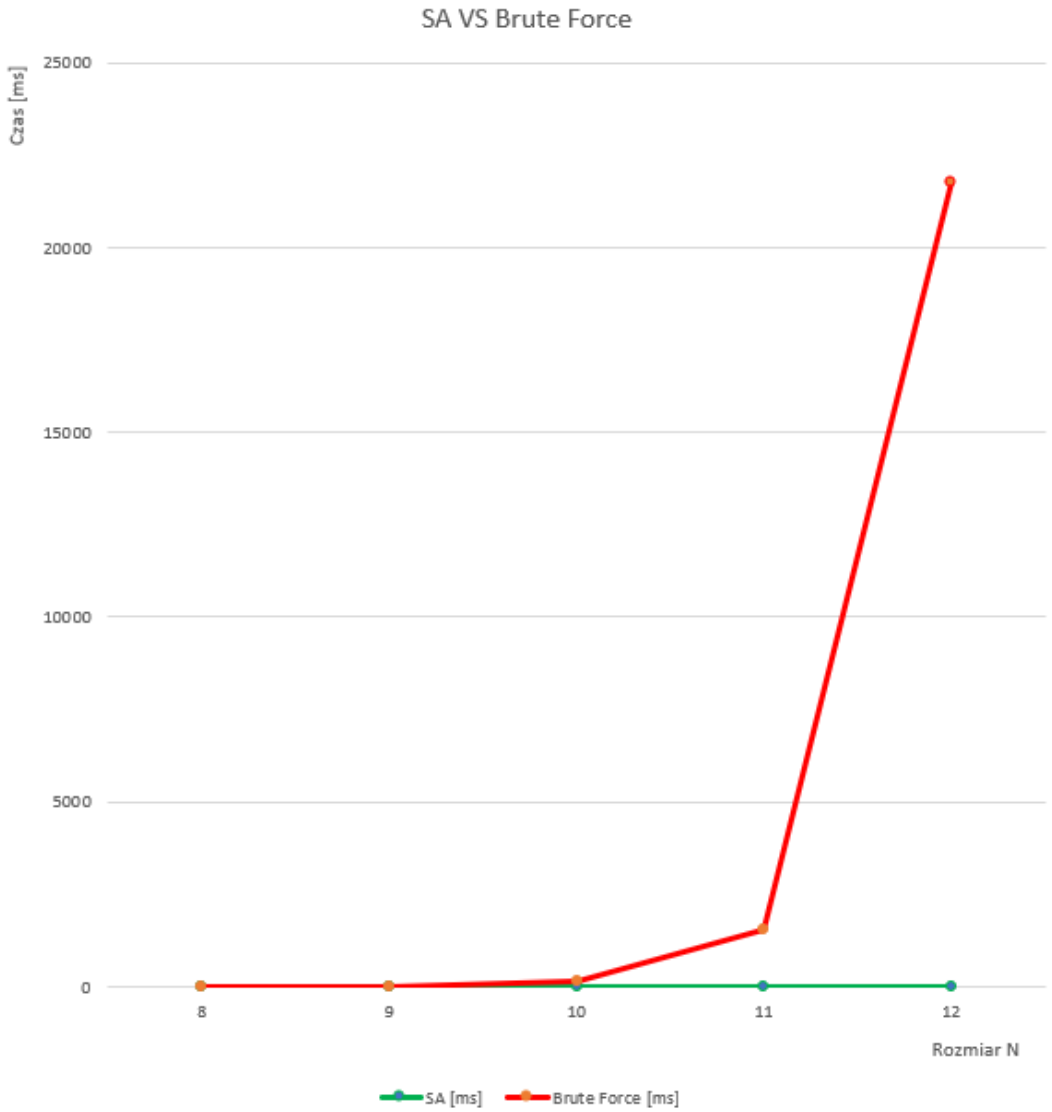
Rysunek 6: Symulowane Wyżarzanie - iteracje

Wyniki pomiarów algorytmu Symulowanego Wyżarzania zostały porównane do algorytmu Brute Force oraz Branch & Bound tylko dla małych instncji ze względu na to, że przeprowadzanie badań na większych rozmiarach dla tego algorytmu zdecydowanie przekroczyłoby dopuszczalny czas oczekiwania na otrzymanie wyniku. Jednak już dla małych instancji problemu mozna łatwo zauważyć różnicę w szybkości działania algorytmów.

Poniższa tabela przedstawia porównanie algorytmu Symulowanego Wyżarzania do algorytmu Brute Force. Wyniki w tabeli zostały przedstawione w milisekundach.

N	SA [ms]	Brute Fore [ms]
8	0.333	1.479
9	0.367	15
10	0.404	142.23
11	0.513	1546.39
12	0.584	21811.19

Tabela 4: Porównanie z Brute Force

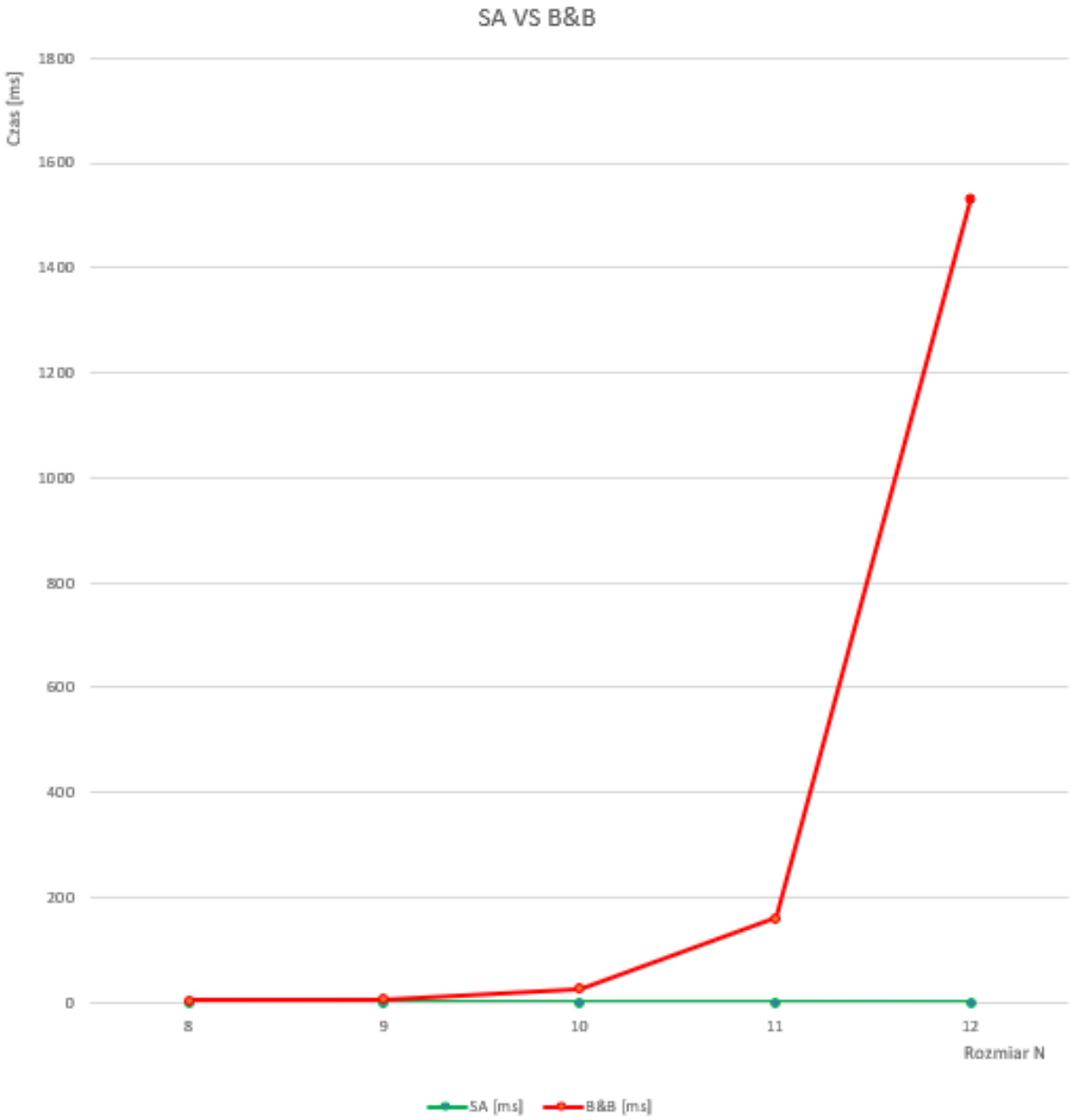


Rysunek 7: SA VS Brute Force

Poniższa tabela przedstawia porównanie algorytmu Symulowanego Wyżarzania do algorytmu Branch & Bound. Wyniki w tabeli zostały przedstawione w milisekundach.

N	SA [ms]	B&B [ms]
8	0.333	4.24
9	0.367	5.806
10	0.404	26.52
11	0.513	160.182
12	0.584	1528.483

Tabela 5: Porównanie z Branch & Bound



Rysunek 8: SA VS B&B

6 Wnioski

Na wykresach można z łatwością zauważyć ogromną różnicę między algorytmem Symulowanego Wyżarzania a dwoma innymi. W przypadku algorytmu Brute Force różnica ta początkowo wynosi lekko ponad 1ms, natomiast już dla instancji o rozmiarze 12 różnica ta to prawie 22 sekundy. Stosując algorytm Symulowango Wyżarzania można zaoszczędzić bardzo dużo czasu, lecz niestety nie zawsze istnieje pewność, że algorytm ten zwrócił najlepsze rozwiązanie, czego nie można zarzucić algorytmowi Brute Force. W przypadku porównania do algorytmu Branch & Bound różnica początkowo wynosi więcej niż dla Brute Force, ponieważ aż 4 ms, jednakże dla o wiele większych instancji różnica ta jest zdecydowanie mniejsza, ponieważ tylko 1.5 sekundy.

Algorytm Symulowanego Wyżarzania jest efektywnym algorytmem optymalizacyjnym, zwłaszcza w przypadkach, gdzie problem ma wiele lokalnych minimów. Jako algorytm heurystyczny nie gwarantuje znalezienia globalnego minimum, ale może dostarczyć dobre przybliżenie. Pozwala zaoszczędzić bardzo dużo czasu, dlatego jest bardziej efektywnym algorytmem niż pozostałe dwa dla problemów o wyższych instancjach. Po dobraniu odpowiednich parametrów pozwala na uzyskanie zadowalających wyników. Algorytm ten jest dobry do stosowania w celu szybkiego znalezienia przybliżonego rozwiązania, jednak nie powinien być stosowany w przypadkach, gdy ważne jest znalezienie prawidłowej odpowiedzi, ponieważ jest wtedy o wiele mniej efektywny niż inne algorytmy. Dobranie parametrów tak, aby być w stanie znaleźć najlepszą odpowiedź prowadzi do wygenerowania bardzo dużej ilości iteracji, w efekcie czego trwa on zdecydowanie za długo oraz nie gwarantuje uzyskania oczekiwanego rezultatu.

7 Kod źródłowy

https://github.com/Klaudiaamarzec/Simulated-Annealing

8 Bibliografia

- http://algorytmy.ency.pl/artykul/symulowane_wyzarzanie
- https://medium.com/@francis.allanah/travelling-salesman-problem-using-simulated-annealing-f547a71ab3c6