Prowadzący: dr inż. Tomasz Kapłon
termin projektu: wt TP, 15.15 – 16.55

Wrocław, 10.11.2016
Weronika Luźna, 209879

Projektowanie efektywnych algorytmów
Zadanie 1.

Algorytm symulowanego wyżarzania dla problemu komiwojażera.

1. Wstęp.

Celem zadania było zaimplementowanie i przetestowanie algorytmu metaheurystycznego (symulowane wyżarzanie) dla problemu komiwojażera.

Problem komiwojażera polega na odnalezieniu minimalnego cyklu Hamiltona w pełnym grafie – przejściu przez n miast jak najkrótszą drogą, odwiedzając każde z nich dokładnie raz. Istnieją dwa rodzaje tego problemu: symetryczny (odległość z punktu A do B jest identyczna jak z B do A) oraz niesymetryczny (odległości są różne w każdą stronę).

Algorytm symulowanego wyżarzania (SA – simulated annealing) opiera się na procesie wziętym ze świata rzeczywistego – powolnym zastyganiu cieczy. Wykorzystywany jest głównie fakt, iż wraz ze spadkiem temperatury cząsteczki mają co raz mniejszą swobodę ruchu, dzięki czemu osiągają dużo bardziej uporządkowany stan wyjściowy niż przy gwałtownym zamrażaniu.

SA służy do optymalizacji - wyszukiwania przybliżonych rozwiązań konkretnych instancji problemów NP-trudnych. Jest rozwinięciem metod iteracyjnych, które opierały się na ciągłym ulepszaniu istniejącego rozwiązania do momentu, gdy nie udawało się go dalej poprawić.
Przejście z jednego rozwiązania do drugiego jest realizowane przez tak zwaną funkcję przejścia i polega na znalezieniu rozwiązania sąsiedniego.. Wadą tych metod było to, że zatrzymywały się one przy rozwiązaniu pseudo-optymalnym stanowiącym jedynie minimum lokalne optymalizowanej funkcji. Algorytm taki nie miał możliwości kontynuacji w kierunku globalnego minimum. SA posiada możliwość wyboru gorszego rozwiązania. Dokonywany jest on z pewnym prawdopodobieństwem, które uzależnione jest bezpośrednio od temperatury. Na początku działania algorytmu temperatura jest wysoka, dzięki czemu konfiguracja rozwiązania ulega wielu zmianom, niejednokrotnie na gorsze. Wraz z kolejnymi iteracjami algorytmu temperatura spada i wybierane są częściej rozwiązania lepsze. Pod koniec temperatura jest na tyle niska, że prawdopodobieństwo wyboru gorszego rozwiązania jest bliskie zeru.

2. Implementacja.

Program powstał przy użyciu języka C++ w środowisku Microsoft Visual Studio 2015. Oparty jest o programowanie zorientowane obiektowo; składa się z trzech klas. Pierwsza z nich: *Matrix*

zawiera macierz, będącą reprezentacją grafu oraz metody potrzebne do wczytywania danych. Druga klasa: *Solution* służy do przechowywania i obsługi rozwiązań (losowanie x0, zamienianie miejscami losowych elementów danego rozwiązania, operator przypisania). Trzecia klasa: *SimAnneal* realizuje algorytm od początku do końca, czyli zawiera:

- inicjalizację parametrów (temperatura początkowa - T0, temperatura końcowa – Tk)

Temperatura początkowa wyznaczana jest automatycznie dla każdej instancji problemu zależnie od jej wielkości i rozrzutu zbioru rozwiązań.

- obliczanie wartości funkcji (długości ścieżki dla danej permutacji)

- obliczanie prawdopodobieństwa na podstawie różnicy między aktualnym rozwiązaniem a rozważanym nowym oraz aktualnej temperatury

- algorytm, który przedstawić można poniższym pseudokodem:

```
wybierz losowo rozwiązanie A
while(T > Tk)
{
for(j od 0 do s)
{
  stwórz rozwiązanie B przez zamianę dwóch losowych elementów w A
  if(wartość(B) < wartość(A)) A = B
  else if(rand(0,1) < P()) A = B
}
T = G()
}
```

Przy czym dla funkcji G() zastosowano schemat geometryczny – przy każdym obrocie pętli while, temperatura maleje α razy. Ustalono temperaturę końcową wynoszącą Tk=0.001.

Parametry zmieniane podczas testów:

- *cooling_rate* (α)= { 0.85, 0.9, 0.95, 0.99, 0.999}

- s = {50, n, $n^2/4$}

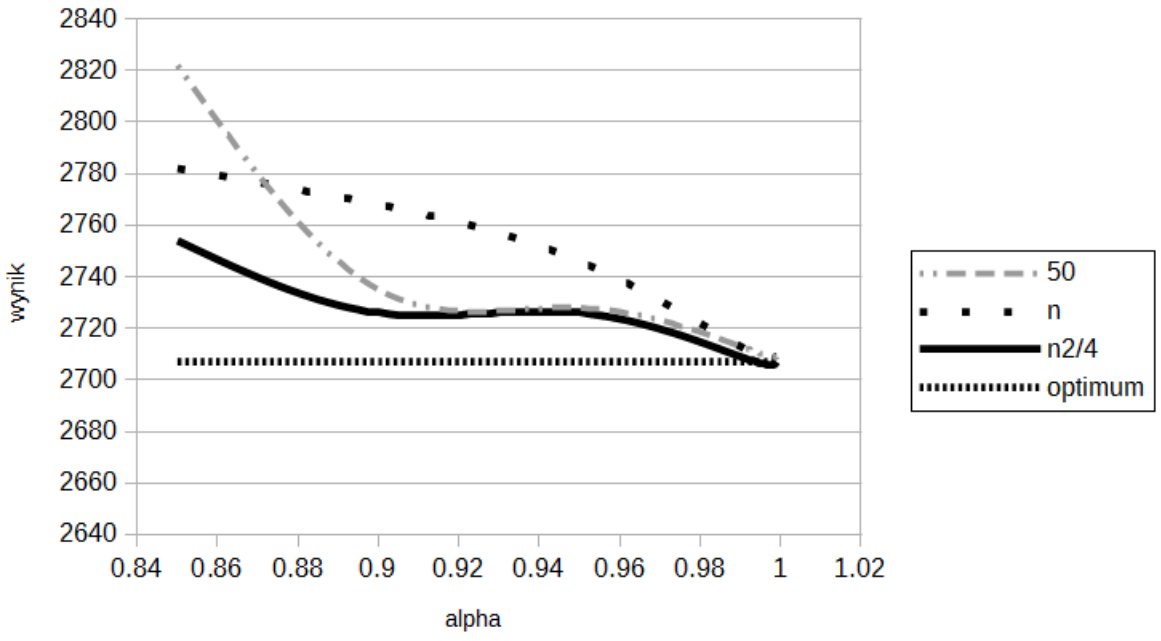
3. Testowanie.

Do testów wybrano trzy instancje o znanych rozwiązaniach optymalnych: (gr21, dantzig42, gr120). Dla każdej z nich zmierzono czas działania algorytmu przy zmiennych parametrach wejściowych. Pomiaru czasu dokonano przy pomocy gotowych funkcji z biblioteki <windows.h>.

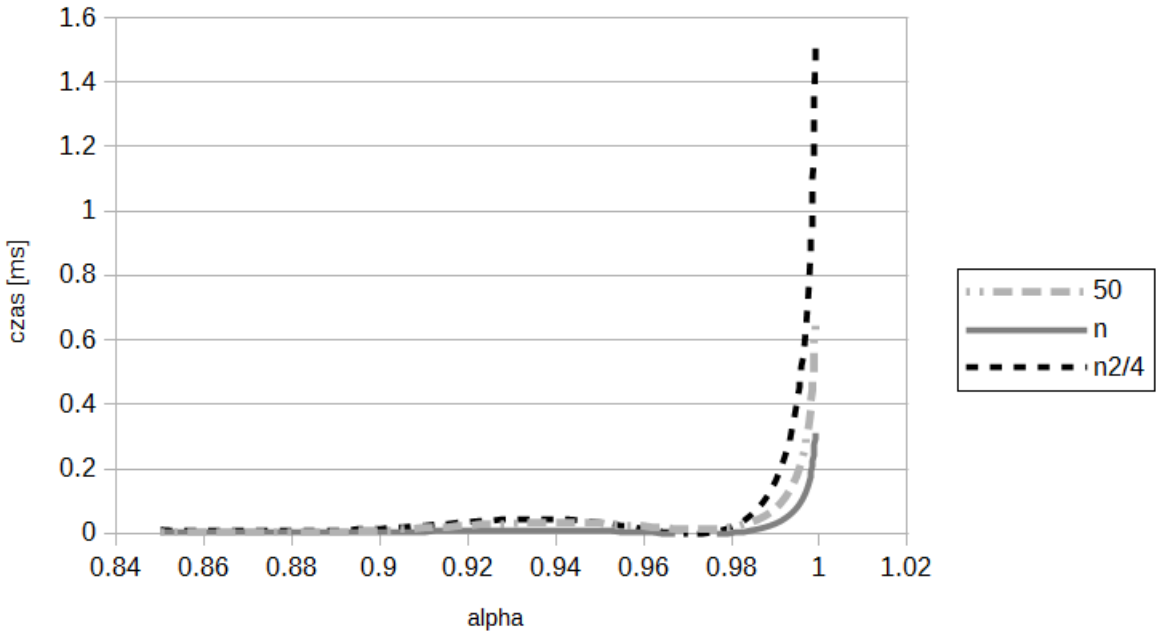
I. Instancja gr21, n=17, wynik optymalny: 2707.

Tabela 1. Zestawienie wyników algorytmu i pomiarów czasu dla gr21.

α / **s**	**0.85**	**0.9**	**0.95**	**0.99**	**0.999**
	WYNIK				
50	2822	2735	2728	2713	2709
n	2782	2768	2746	2713	2709
$n^2/4$	2754	2726	2726	2709	2707
	CZAS [ms]				
50	0.0058	0.01008	0.03166	0.07860	0.65982
n	0.0029	0.00469	0.00805	0.03097	0.31113
$n^2/4$	0.0119	0.01596	0.03511	0.16324	1.51878



Wykres 1. Zależność wyników algorytmu od współczynnika alpha dla różnych ilości iteracji.

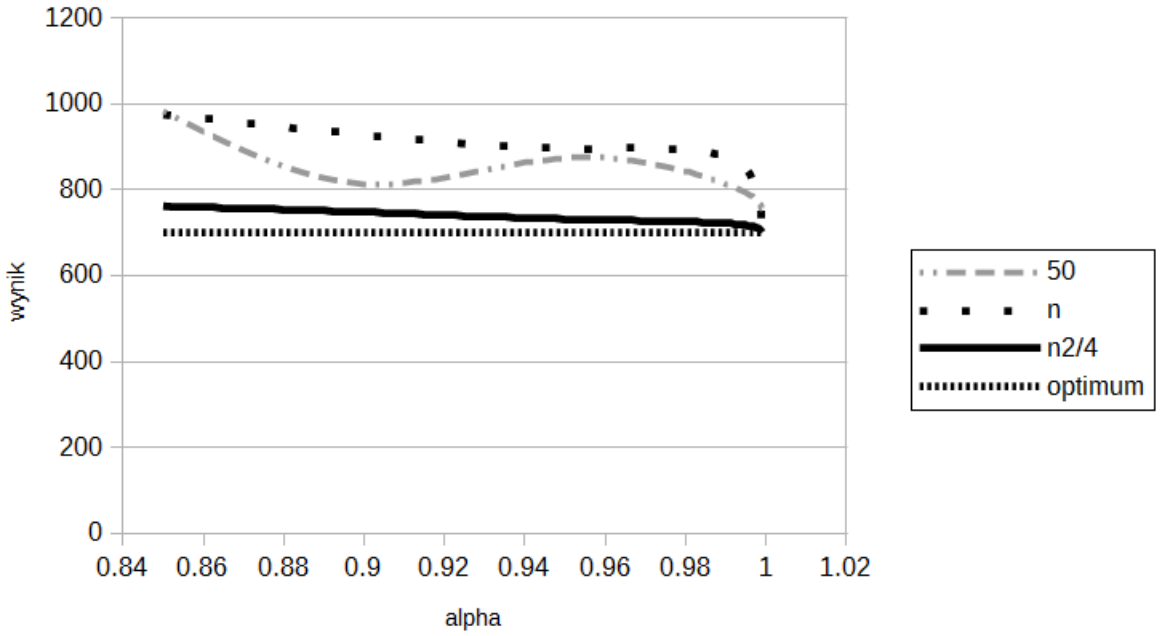


Wykres 2. Zależność czasu działania od współczynnika alpha dla różnych ilości iteracji.

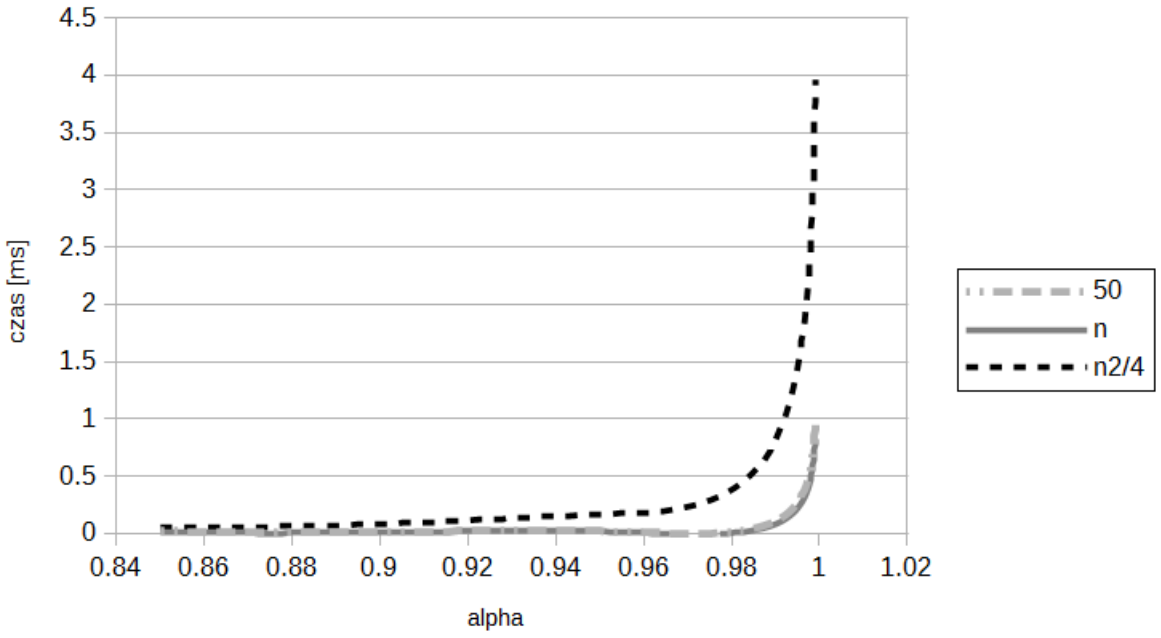
II. Instancja dantzig42, n=42, wynik optymalny: 699.

Tabela 2. Zestawienie wyników algorytmu i pomiarów czasu dla dantzig42.

α / **s**	**0.85**	**0.9**	**0.95**	**0.99**	**0.999**
	WYNIK				
50	983	814	874	813	756
n	975	928	896	877	720
$n^2/4$	762	748	732	722	701
	CZAS [ms]				
50	0.02108	0.01140	0.02023	0.10864	0.96183
n	0.00714	0.00979	0.01973	0.08144	0.82406
$n^2/4$	0.05267	0.08224	0.16712	0.81699	3.96136



Wykres 3. Zależność wyników algorytmu od współczynnika alpha dla różnych ilości iteracji.

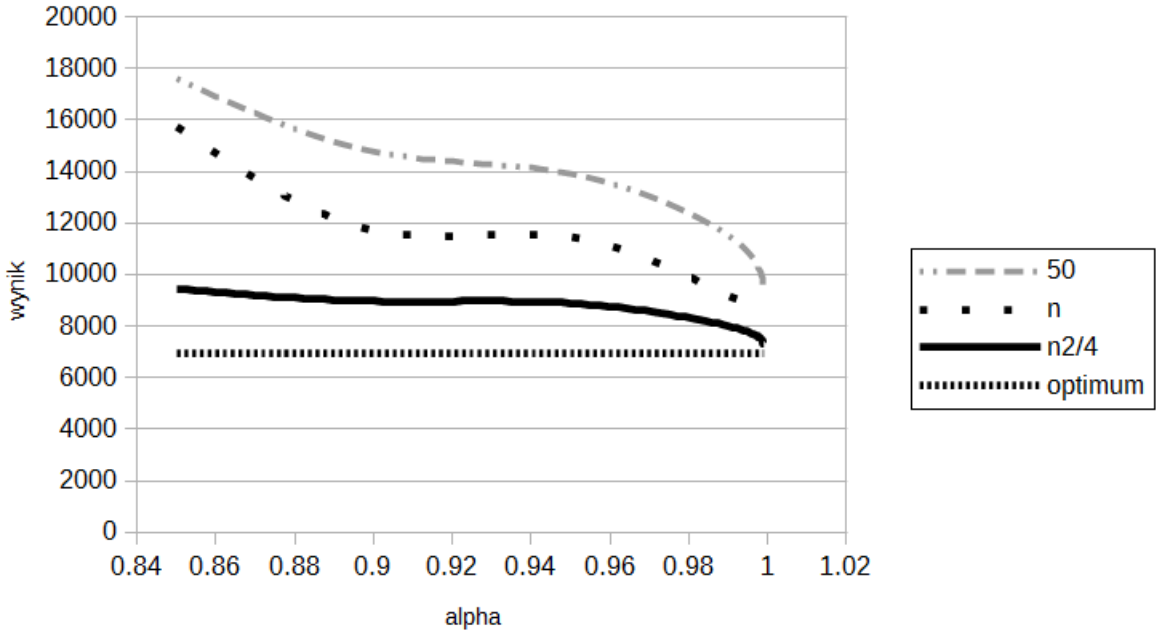


Wykres 4. Zależność czasu działania od współczynnika alpha dla różnych ilości iteracji.

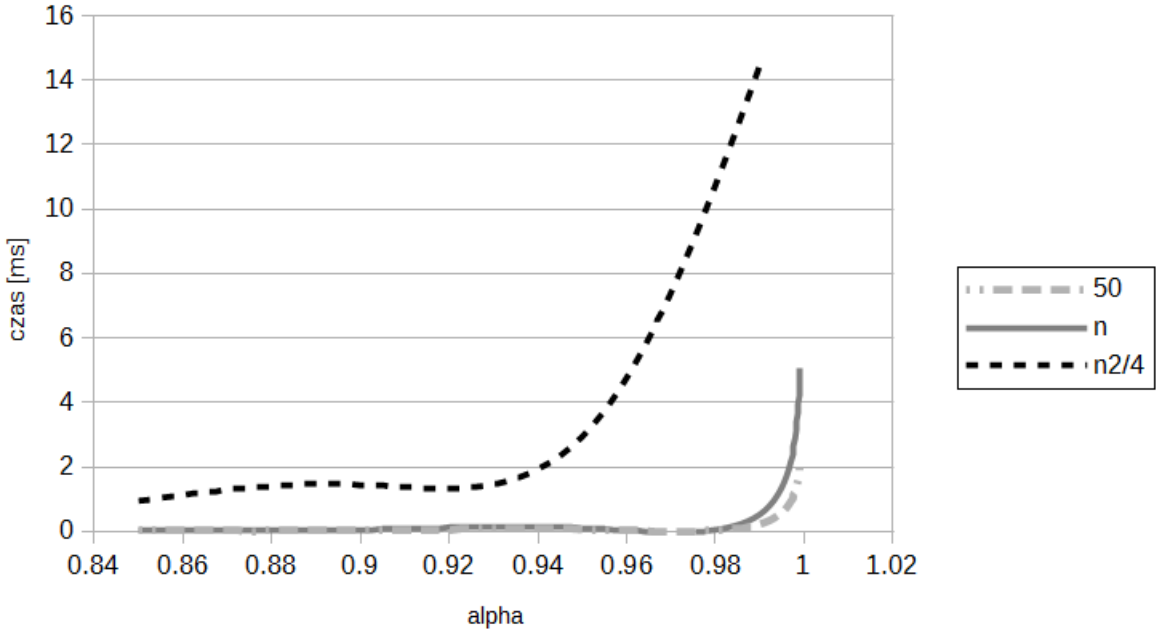
III. Instancja gr120, n=120, wynik optymalny: 6942.

Tabela 3. Zestawienie wyników algorytmu i pomiarów czasu dla gr120.

α / s	0.85	0.9	0.95	0.99	0.999
	WYNIK				
50	17603	14771	13910	11517	9399
n	15741	11716	11462	9184	9266
$n^2/4$	9443	8974	8891	8007	7291
	CZAS [ms]				
50	0.02399	0.03120	0.06527	0.23087	2.19568
n	0.04215	0.06079	0.10980	0.53199	5.07152
$n^2/4$	0.95544	1.45688	2.94819	14.4669	92.8473+-



Wykres 5. Zależność wyników algorytmu od współczynnika alpha dla różnych ilości iteracji.



Wykres 6. Zależność czasu działania od współczynnika alpha dla różnych ilości iteracji.

4. Wnioski.

W trakcie realizacji projektu zauważono, że dobór parametrów wejściowych dla algorytmu ma ogromny wpływ na jakość zwracanego rozwiązania, a także czas potrzebny do jego obliczenia.

Jak wynika z wykresów 1, 3 i 5, wraz ze zwiększaniem parametru α, wartości malały przybliżając się co raz bardziej do pożądanego optimum. Niestety udało się je osiągnąć tylko raz (dla gr21), co z kolei nasuwa wniosek, że symulowane wyżarzanie działa lepiej dla mniejszych instancji.

Dodatkowo zdarzały się skoki w wielkościach rozwiązań, na przykład dla s=50 w drugiej instancji (wykres 3.) widać wyraźnie, że krzywa jest malejąca, ale mniej więcej w jej środkowej części pojawia się wartość nieoczekiwana - obliczono dużo gorsze rozwiązanie niż dla poprzednich α. Może to wynikać z niedoskonałości samej implementacji lub specyfiki danych wejściowych.

Czas wykonywania się algorytmu wzrastał razem z liczbą instancji, co widać na wykresach 2,4 i 6. Porównując te trzy wykresy, zauważyć można również, że ogółem czas rozwiązania zadania wydłużał się wraz ze wzrostem współczynnika α**.** Dużą rolę odgrywała także liczba iteracji. Jej zwiększanie pozwalało na znalezienie lepszego optimum w każdym przypadku, ale algorytm działał

wtedy odpowiednio dłużej.

Podsumowując, najlepszy wynik osiągnięto dla małej instancji, dla największej wartości parametru α – bliskiej 1, przy największej ilości iteracji.

Najważniejsza dostrzeżona zależność: udawało się uzyskiwać albo krótsze czasy, albo lepsze rozwiązania. Zysk pod jednym względem generował stratę pod tym drugim.