Projektowanie Efektywnych Algorytmów

Kierunek *Informatyka*	**Termin** *Poniedziałek 15:25*
Temat *Tabu search i symulowane wyżarzanie*	**Problem** *TSP*
Skład grupy *241385 Radosław Lis*	**Nr grupy** *-*
Prowadzący *Mgr inż. Radosław Idzikowski*	**data** *17 grudnia 2019*

Spis treści

1 Opis problemu

Problem komiwojażera (ang. **TSP** - *Travelling Salesman Problem*) to jedno z najbardziej powszechnych zagadnień z dziedziny algorytmiki. W celu zobrazowania zagadnienia, należy wyobrazić sobie komiwojażera, który podróżuje między miastami w prowincji, sprzedając swoje towary. Wyrusza ze swojego domu, po czym jego trasa przebiega dokładnie jeden raz przez każde miasto w prowincji, aż na końcu wraca do domu rodzinnego. Rozwiązanie problemu to znalezienie odpowiedniej drogi i kosztów podróży (odległość, czas itp.), która maksymalnie je zminimalizuje.

Z matematycznej perspektywy wygląda to tak, że miasta są wierzchołkami grafu, a łączace je trasy to krawędzie z odpowiednimi wagami. Jest to graf pełny, ważony oraz może być skierowany - co tworzy problem *asymetryczny*. Rozwiązanie problemu komiwojażera sprowadza się do znalezienia właściwego - o najmniejszej sumie wag krawędzi - *cyklu Hamiltona*, czyli cyklu przechodzącego przez każdy wierzchołek grafu dokładnie jeden raz. Przeszukanie wszystkich cykli (czyli zastosowanie metody *Brute Force*) nie jest optymalną metodą, jako że prowadzi do wykładniczej złożoności obliczeniowej - $O(n!)$, dla której problemy o dużym n są traktowane jako nierozwiązywalne. Klasyfikuje to problem komiwojażera jako *problem NP-trudny*, czyli niedający rozwiązania w czasie wielomianowym. To powoduje konieczność skorzystania z tzw. *algorytmów heurystycznych* bądź *metaheurystycznych* (bardziej ogólnych), a w naszym przypadku konkretnie algorytmów przeszukiwania lokalnego - *Tabu search* oraz *symulowanego wyżarzania*.

2 Metoda rozwiązania

2.1 Tabu search

Algorytm *Tabu search* (przeszukiwanie tabu, poszukiwanie z zakazami) to jedna z metod przeszukiwania lokalnego bazująca na dynamicznej zmianie sąsiedztwa danego rozwiązania i szukaniu lokalnie najlepszych rozwiązań, przeznaczona do rozwiązywania problemów optymalizacyjnych. Przeszukiwanie, dzięki wielu parametrom cechującym *Tabu search*, może - choć nie musi - doprowadzić do otrzymania globalnie najlepszego rozwiązania. Algorytm charakteryzuje znikoma złożoność pamięciowa oraz brak jawnie zdefiniowanej czasowej złożoności obliczeniowej, gdyż algorytm kończy się wraz z pewnym warunkiem, w naszym przypadku wykonywanie go trwa określony czas.

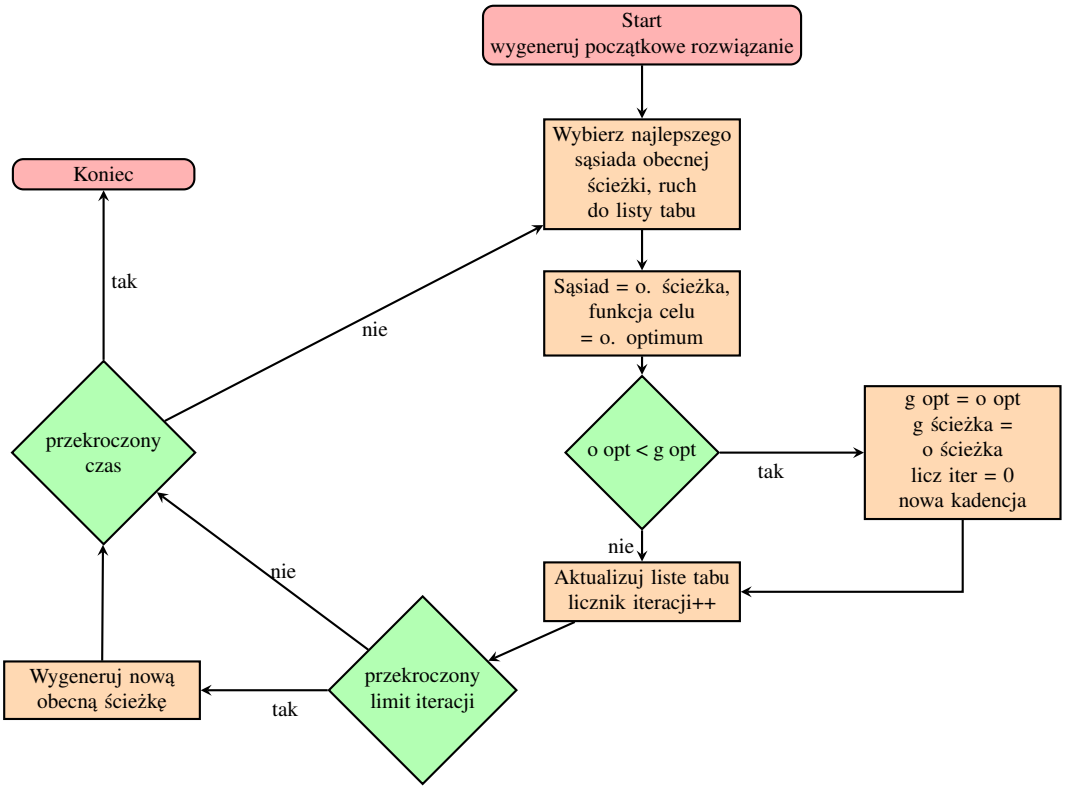
2.1.1 Parametry zaimplementowanego Tabu search

- **czas** - czas działania algorytmu,
- **globalnie najlepsza ścieżka** - ścieżka, która dotychczasowo dała najmniejszą funkcję celu,
- **obecna ścieżka** - obecnie eksplorowana ścieżka,
- **globalne optimum** - funkcja celu globalnie najlepszej ścieżki,
- **obecne optimum** - funkcja celu obecnie eksplorowanej ścieżki,
- **ruch** - ruch prowadzący z obecnej do sąsiedniej ścieżki (dalej *sąsiada*),
- **najlepszy sąsiad** - sąsiednia ścieżka, która daje najmniejszą funkcję celu z całego sąsiedztwa,
- **lista tabu** - lista zawierająca wykonane ostatnio ruchy, przechowywane przez daną liczbę iteracji, tzw. *kadencję*,
- **kadencja** - liczba iteracji przez jaką ruch jest przechowywany na liście tabu,
- **limit iteracji** - limit iteracji bez poprawy globalnego optimum, po osiągnięciu którego jest generowana nowa ścieżka,
- **licznik iteracji** - licznik iteracji w zakresie każdej wygenerowanej na nowo ścieżki,
- **dzielnik kadencji** - liczba przez którą dzielona jest kadencja w razie znalezienia globalnego optimum (*intensyfikacja*),

- **liczba losowych wierzchołków** - liczba początkowych wierzchołków, która jest losowana przy generowaniu nowej ścieżki,
- **algorytm** - typ algorytmu, który generuje nową ścieżkę,
- **rodzaj sąsiedztwa** - ruch - *swap*, *reverse* bądź *insert* - definiujący powstawanie sąsiadów,

2.1.2 Opis i schemat działania algorytmu

Działanie algorytmu rozpoczyna się od uruchomienia stopera wygenerowania początkowej ścieżki, za pomocą jednego z dwóch **algorytmów** - zachłannego bądź algorytmu, polegającego na redukcji macierzy danej instancji, który został zaimplementowany przy implementacji algorytmu *Branch&Bound* w poprzednim etapie projektowym. Początkowa ścieżka staje się zarówną **obecną ścieżką** jak i - najprawdopodobniej tymczasowo - **globalnie najlepszą ścieżką**, analogicznie jej funkcja celu staje się zarówno **obecnym**, jak i **globalnym optimum**. Następnie przeszukiwane jest całe sąsiedztwo, za pomocą **ruchów** o danym **rodzaju sąsiedztwa**, z których zostaje zapamiętany ruch prowadzący do **najlepszego sąsiada** i nienależący do **listy tabu**, chyba że spełnia tzw. *kryterium aspiracji*, czyli prowadzi do uzyskania **globalnego optimum**. Po przeszukaniu zostaje wykonany zapamiętany ruch i nowa ścieżka zostaje **obecną ścieżką**, a jej funkcja celu **obecnym optimum** oraz jeśli okazała się mniejsza niż **globalne optimum** to jest ono aktualizowane wraz z **globalnie najlepszą ścieżką**, a **licznik iteracji** zostaje zresetowany i kadencja dla przyszłych ruchów jest dzielona przez **dzielnik kadencji**. Wszystkie „pozostałe iteracje" ruchów na **liście tabu** zostają zdekrementowane, a te których jest równa 0 są z niej usuwane. Ruch zostaje wrzucony na **listę tabu** wraz z obecną **kadencją**, a **licznik iteracji** jest inkrementowany. Sprawdzane jest czy **licznik iteracji** nie przekroczył **limitu iteracji**, jeśli tak to generowana jest - danym **algorytmem** - nowa ścieżka wraz z odpowiednią **liczbą początkowych losowych wierzchołków**, która staje się **obecną ścieżką**. Zatrzymywany jest stoper i nastepuje sprawdzenie czy nie został przekroczony czas przeznaczony na działanie algorytmu, jeśli tak to następuje koniec działania algorytmu. Następuje powrót do przeszukiwania sąsiedztwa **obecnej ścieżki**.



Rysunek 1: Schemat *Tabu search*

2.1.3 Macierz reprezentująca przykładowy problem komiwojażera

	0	1	2	3	4	5
0	∞	81	50	18	75	39
1	81	∞	76	21	37	26
2	50	76	∞	24	14	58
3	18	21	24	∞	19	58
4	75	37	14	19	∞	31
5	39	26	58	58	31	∞

Początkowa ścieżka wygenerowana algorytmem redukcyjnym π = <0,3,4,2,5,1,0>, koszt ścieżki to 216.

2.1.4 Rodzaje sąsiedztw

W ramach implementacji zostały zdefiniowane 3 podstawowe typy sąsiedztwa (*a,b* to indeksy wierzchołków *i,j)*:

1. **swap (i, j)** - zamiana miejscami wierzchołka *i* z *j*:

   swap(4,1): <0,3,4,2,5,1,0> $\rightarrow$ <0,3,1,2,5,4,0>

   Koszt sprowadza się do odjęcia czterech krawędzi:

   matrix[*route[a-1]*][*route[a]*]
   matrix[*route[a]*][*route[a+1]*]
   matrix[*route[b-1]*][*route[b]*]
   matrix[*route[b]*][*route[b+1]*]

   Dodania czterech krawędzi:

   matrix[*route[a-1]*][*route[b]*]
   matrix[*route[b]*][*route[a+1]*]
   matrix[*route[b-1]*][*route[a]*]
   matrix[*route[a]*][*route[b+1]*]

2. **reverse (i, j)** - odwrócenie kolejności między wierzchołkami *i* oraz *j* włącznie:

   reverse(4,1): <0,3,4,2,5,1,0> $\rightarrow$ <0,3,1,5,2,4,0>

   Koszt sprowadza się do odjęcia dwóch krawędzi:

   matrix[*route[a-1]*][*route[a]*]
   matrix[*route[b]*][*route[b+1]*]

   Dodania dwóch krawędzi:

   matrix[*route[a-1]*][*route[b]*]
   matrix[*route[a]*][*route[b+1]*]

   Oraz (na potrzeby problemów asymetrycznych) do $(b-a)$-krotnego zastąpienia krawędzi (a++):

   matrix[*route[a]*][*route[a+1]*] krawędzią matrix[*route[a+1]*][*route[a]*]

3. **insert (i, j)** - przesunięcie wierzchołka *i* na pozycję *j-tą* i odpowiednie, ewentualne przesunięcie pozostałych wierzchołków:

$$\text{insert}(4,1): \quad \langle 0,3,4,2,5,1,0\rangle \rightarrow \langle 0,4,3,2,5,1,0\rangle$$

Koszt sprowadza się do odjęcia trzech krawędzi:

matrix[*route[a]*][*route[a+1]*]
matrix[*route[b-1]*][*route[b]*]
matrix[*route[a-1]*][*route[a]*]

Oraz dodania trzech krawędzi:

matrix[*route[a-1]*][*route[a+1]*]
matrix[*route[b-1]*][*route[a]*]
matrix[*route[a]*][*route[b]*]

2.1.5 Kilka pierwszych iteracji dla danej instancji

Dla początkowej ścieżki $\pi = \langle 0,3,4,2,5,1,0\rangle$ o funkcji celu 216 przeszukiwane jest (metodą *swap*) całe jej sąsiedztwo. Najlepszy ruch, czyli zamiana wierzchołków 5 i 1 generuje nam nową ścieżkę $\pi = \langle 0,3,4,2,1,5,0\rangle$, której funkcja celu wynosi już 192, które staje się globalnym optimum, a utworzona ścieżka najlepszą globalnie ścieżką oraz resetowany jest licznik iteracji i dochodzi do *intensyfikacji*. Na naszej liście tabu - dotychczas pustej - znajduje się już pierwszy ruch, czyli zamiana wierzchołków 1,5 o pozostałym „okresie życia na liście" równym kadencji. Następnie przeszukiwane jest sąsiedztwo ścieżki $\pi = \langle 0,3,4,2,1,5,0\rangle$, z którego najlepszym ruchem jest swap wierzchołków 4 i 2, który daje nam nową ścieżkę $\pi = \langle 0,3,2,4,1,5,0\rangle$. Jej funkcja celu to 158, co daje już potężne ulepszenie w stosunku do początkowego rozwiązania. Resetowany jest licznik iteracji, nadpisywane jest globalne optimum i ścieżka, a ruch jest dodany do tabu i następuje teraz przeszukanie sąsiedztwa ścieżki $\pi = \langle 0,3,2,4,1,5,0\rangle$. Najlepszy sąsiad powstaje przez zamianę wierzchołków 3 i 5, ale już nie daje globalnie najlepszego rozwiązania, gdyż funkcja celu nowej ścieżki wynosi 187. Ruch ponownie jest dodawany, ale tym razem już licznik iteracji nie jest resetowany, tylko inkrementowany, gdyż nie zostało znalezione najlepsze globalnie rozwiązanie.

Cała procedura trwa aż do przekroczenia czasu przeznaczonego na działanie algorytmu, a po każdym przekroczeniu limitu iteracji jest generowane nowe rozwiązanie, które staje się obecną ścieżką. W tym przypadku 158 okazuje się być na koniec optymalnym rozwiązaniem, ale zazwyczaj *Tabu search* nie wykazuje się aż taką efektywnością i na rozwiązanie trzeba „poczekać" więcej niż dwie iteracje.

2.1.6 Testy

Instancja	Alias	Kadencja	Czas	Limit iteracji	Dzielnik kadencji	Liczba losowych wierzcholkow	Algorytm	Sąsiedztwo	Rozwiązanie	Najlepszy rezultat	Błąd względny
gr17	data17	*10*	*1*	*5000*	*4*	*1*	*true*	*0*	2085	2085	0,00%
gr21	data21	*10*	*1*	*5000*	*4*	*1*	*true*	*0*	2707	2707	0,00%
gr24	data24	*10*	*1*	*5000*	*4*	*1*	*true*	*0*	1272	1272	0,00%
fri26	data26	*10*	*1*	*5000*	*4*	*1*	*true*	*0*	937	937	0,00%
bayg29	data29	*10*	*1*	*5000*	*4*	*1*	*true*	*0*	1610	1610	0,00%
dantzig42	data42	*10*	*5*	*5000*	*4*	*1*	*true*	*0*	699	699	0,00%
brazil58	data58	*10*	*5*	*5000*	*4*	*1*	*true*	*0*	25395	25395	0,00%
gr120	data120	*35*	*120*	*5000*	*4*	*2*	*true*	*0*	6942	6947	0,07%
swiss42	swiss42	*10*	*1*	*5000*	*4*	*1*	*true*	*0*	1273	1273	0,00%

Tablica 1: Tabela najlepszych ustawień parametrów *Tabu* dla danych instancji symetrycznych (TSP) wraz z najlepszym znalezionym rozwiązaniem

Instancja	Alias	Kadencja	Czas	Limit iteracji	Dzielnik kadencji	Liczba losowych wierzcholkow	Algorytm	Sąsiedztwo	Rozwiązanie	Najlepszy rezultat	Błąd względny
br17	data17v2	*10*	*1*	*5000*	*4*	*1*	*true*	*0*	39	39	0,00%
ftv33	data34	*22*	*45*	*5000*	*4*	*2*	*false*	*1*	1286	1286	0,00%
ftv35	data36	*20*	*25*	*5000*	*4*	*2*	*false*	*1*	1473	1473	0,00%
ftv38	data39	*20*	*60*	*5000*	*4*	*2*	*false*	*1*	1530	1530	0,00%
p43	data43	*20*	*10*	*5000*	*4*	*1*	*true*	*0*	5620	5620	0,00%
ftv44	data45	*25*	*15*	*5000*	*4*	*1*	*true*	*2*	1613	1615	0,12%
ry48p	data48	*30*	*30*	*5000*	*4*	*1*	*true*	*0*	14422	14422	0,00%
ftv53	data53	*50*	*45*	*5000*	*4*	*1*	*true*	*2*	6905	7068	2,36%
ftv55	data56	*45*	*60*	*5000*	*4*	*2*	*false*	*2*	1608	1658	3,11%
ftv64	data65	*45*	*30*	*5000*	*4*	*2*	*true*	*2*	1839	1875	1,96%
ft70	data70	*55*	*240*	*5000*	*4*	*1*	*false*	*2*	38673	38817	0,37%
ftv70	data71	*50*	*60*	*5000*	*7*	*1*	*true*	*1*	1950	1984	1,74%
kro124	data100	*80*	*45*	*5000*	*13*	*1*	*false*	*2*	36230	38230	5,52%
ftv170	data171	*120*	*60*	*5000*	*9*	*5*	*false*	*2*	2755	3121	13,28%
rbg323	data323	*80*	*5*	*5000*	*4*	*8*	*true*	*2*	1326	1440	8,60%
rbg358	data358	*80*	*5*	*5000*	*4*	*0*	*true*	*2*	1163	1328	14,19%
rbg403	data403	*80*	*5*	*5000*	*4*	*0*	*true*	*2*	2465	2496	1,26%
rbg443	data443	*80*	*5*	*5000*	*4*	*0*	*true*	*2*	2720	2788	2,50%
ftv47	ftv47	*33*	*240*	*5000*	*4*	*2*	*false*	*1*	1776	1776	0,00%

Tablica 2: Tabela najlepszych ustawień parametrów *Tabu* dla danych instancji asymetrycznych (ATSP) wraz z najlepszym znalezionym rozwiązaniem

2.1.6.1 Pierwsza faza testów

W tablicach 1 i 2 zostały zaprezentowane ustawienia parametrów dla danych instancji, które pozwoliły uzyskać możliwie najbliższe rozwiązaniu wyniki. Pierwsza faza testów polegała na testowaniu różnych ustawień parametrów *Tabu* za pomocą zagnieżdżonych pętli. Dla każdej kombinacji parametrów zostało przeprowadzonych tylko 10 pomiarów, ze względu na dość dużą ilość czasu potrzebną na przetestowanie wszystkich kombinacji.

Wartości *true* i *false* w parametrze *Algorytm* oznaczają odpowiednio skorzystanie z losowo-zachłannego i losowo-redukcyjnego algorytmu po napotkaniu zdarzenia krytycznego i przekroczeniu limitu iteracji bez poprawy. Zaś jeśli chodzi o parametr *Sąsiedztwo* to wartości 0,1 i 2 oznaczają odpowiednio uzyskiwanie sąsiedztwa po zamianach typu *reverse*, *swap* oraz *insert*.

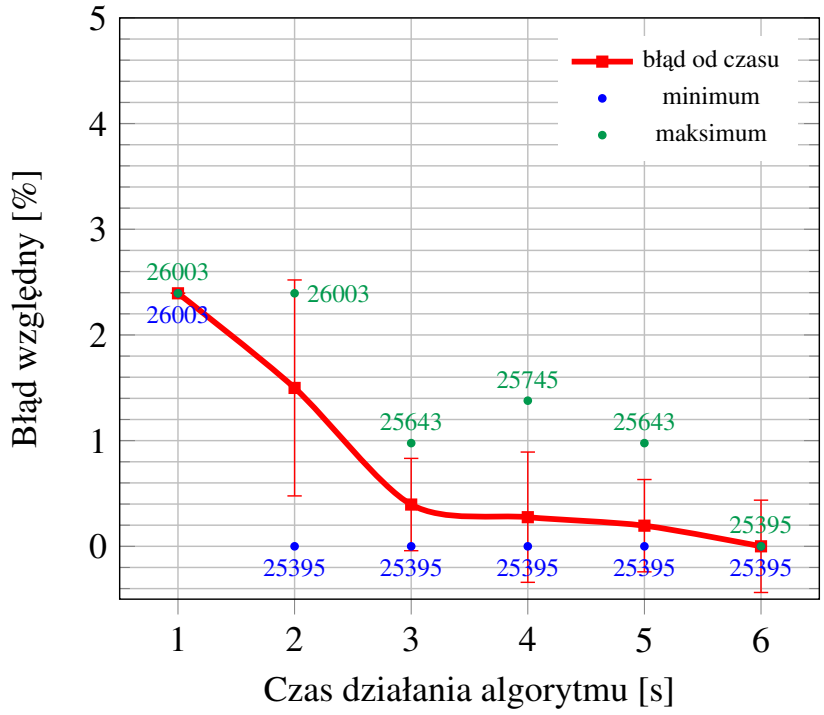
2.1.6.2 Druga faza testów

Druga faza testów polegała na wielokrotnym (100 iteracji) wykonaniu algorytmu dla znalezionej wcześniej, najlepszej kombinacji parametrów dla danej instancji. Najmniejsze-najlepsze rozwiązanie zostało wpisane w kolumnę *Najlepszy rezultat* w tablicach 1 i 2.

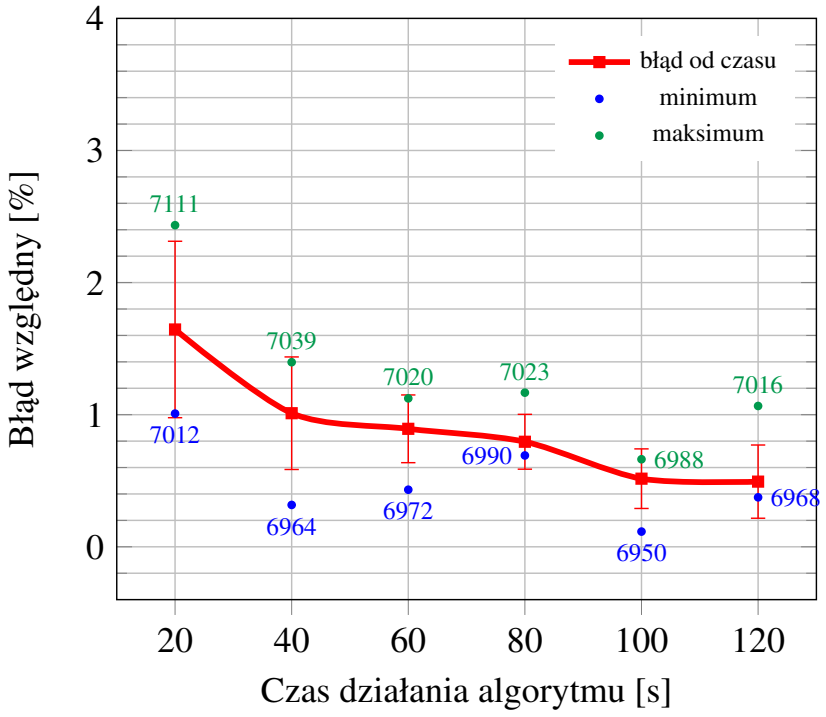
2.1.6.3 Trzecia faza testów

W ramach trzeciej fazy testów zostało wybranych 11 instancji - *brazil58*, *gr120*, *ftv33*, *ftv35*, *ftv38*, *p43*, *ry48p*, *ftv53*, *ftv55*, *ftv70* oraz *ftv170* - na których zostały przeprowadzone testy, których celem było zobrazowanie zależności średniego błędu względnego od czasu działania algorytmu. Dla każdej instancji zostało wybranych - w równych odstępach - pięć lub sześć różnych czasów działania, dla których - wraz z optymalnymi parametrami dla danej instancji - zostało przeprowadzone po 5 testów, a średnie wyniki - wraz z odchyleniami standardowymi - zostały ukazane na poniższych wykresach. Zielone kropki na wykresach oznaczają maksymalne (najgorsze) rozwiązanie znalezione dla danego czasu, a niebieskie minimalne (najlepsze).

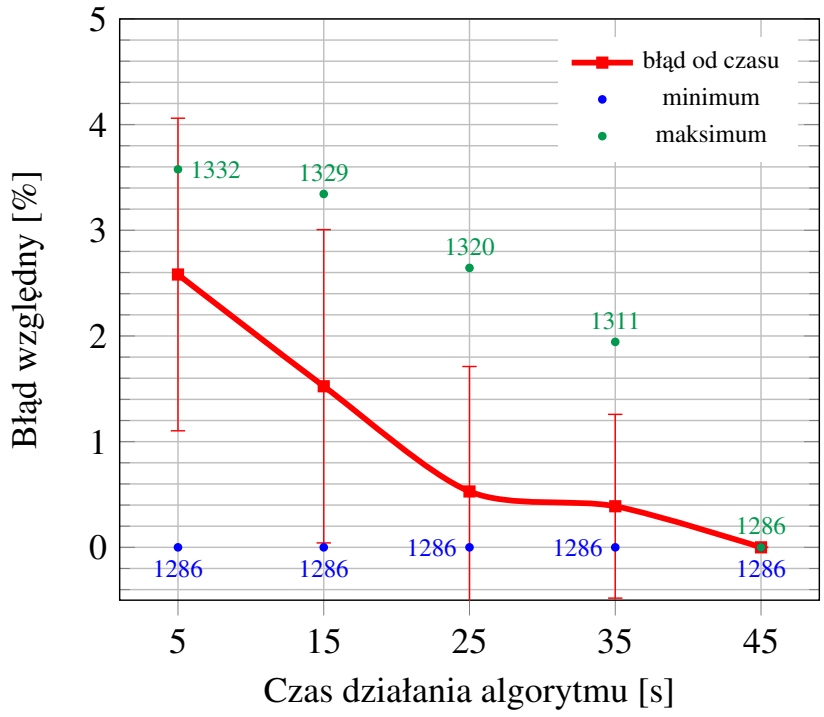
2.1.7 Wyniki testów



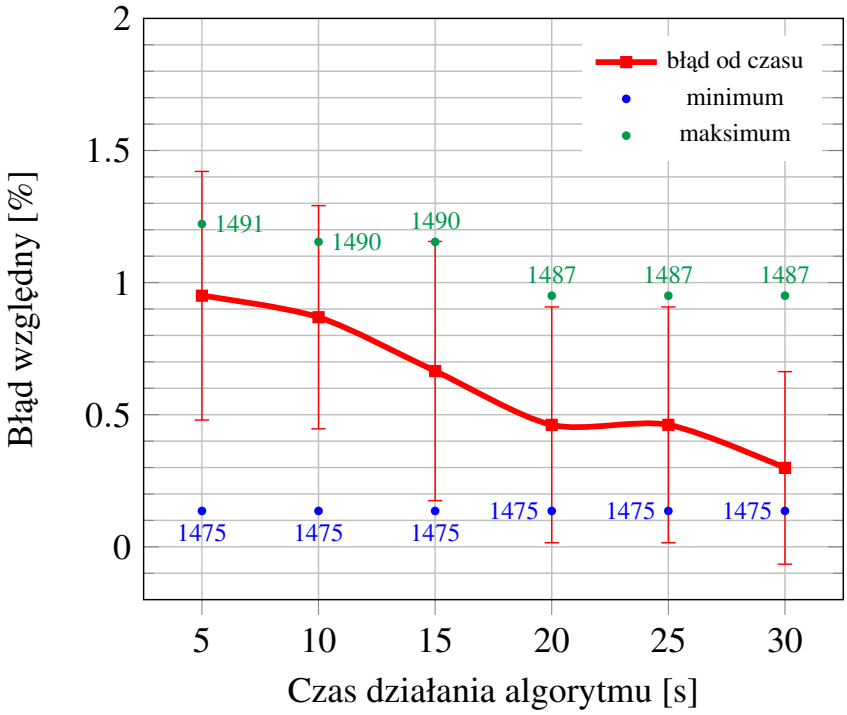
Rysunek 2: Wyniki działania algorytmu *Tabu search* dla instancji *brazil58.tsp*



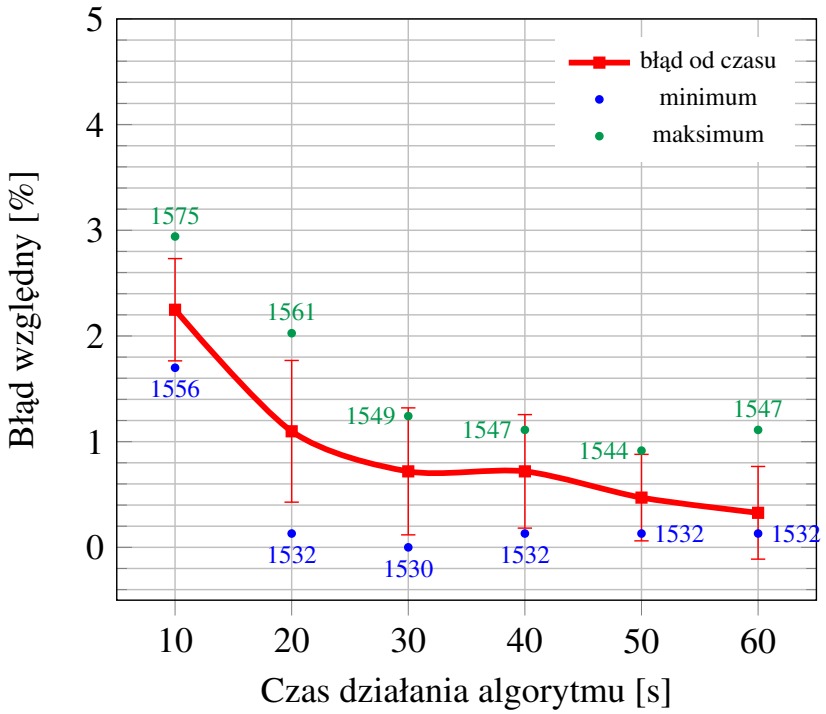
Rysunek 3: Wyniki działania algorytmu *Tabu search* dla instancji *gr120.tsp*



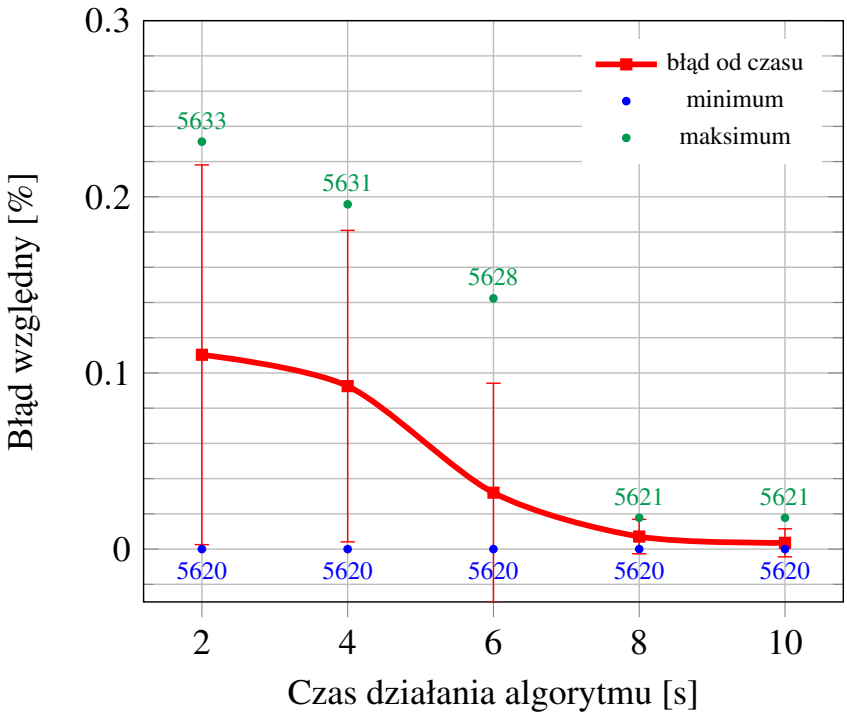
Rysunek 4: Wyniki działania algorytmu *Tabu search* dla instancji *ftv33.atsp*



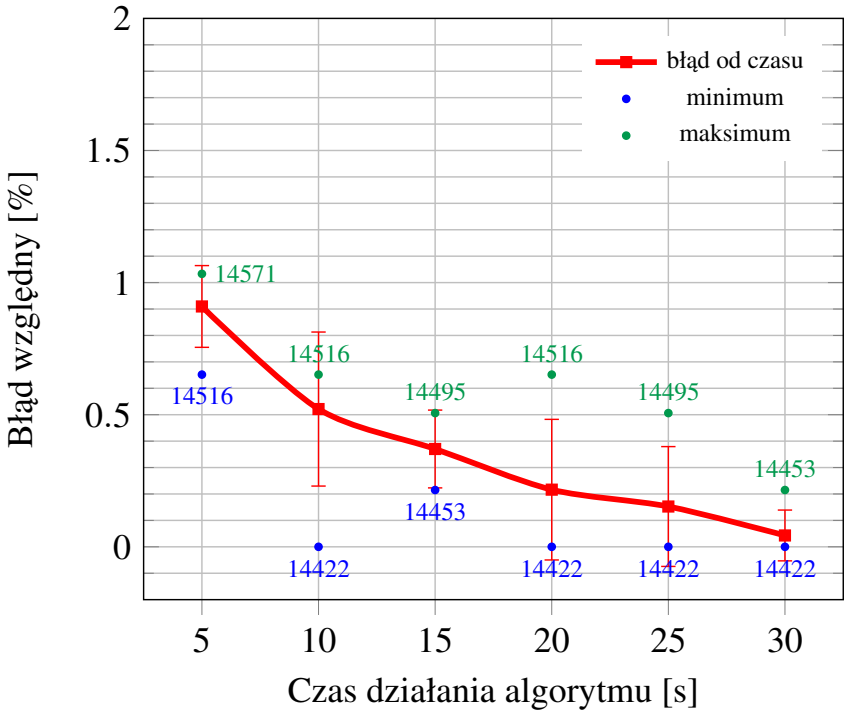
Rysunek 5: Wyniki działania algorytmu *Tabu search* dla instancji *ftv35.atsp*



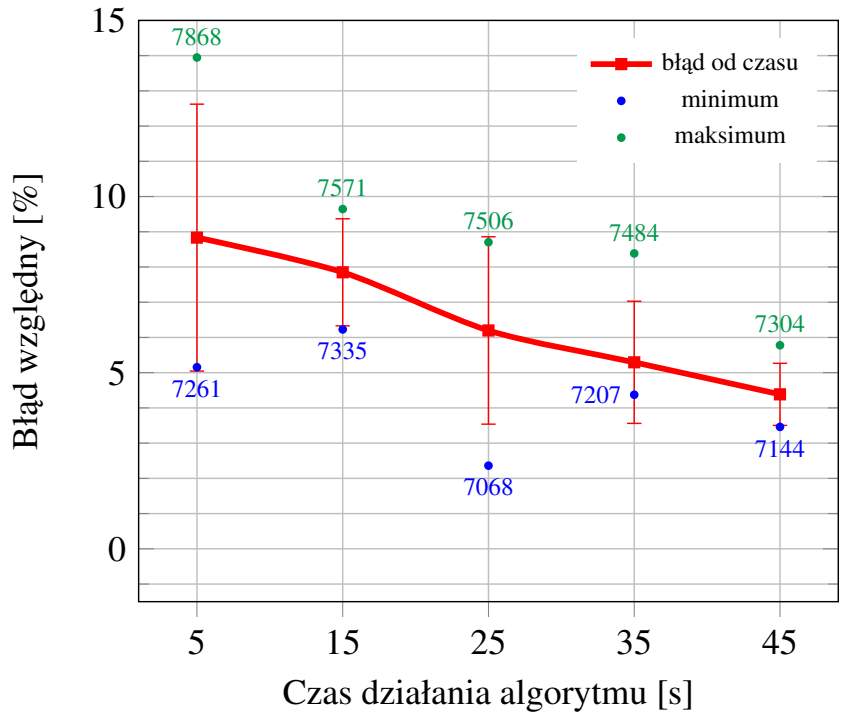
Rysunek 6: Wyniki działania algorytmu *Tabu search* dla instancji *ftv38.atsp*



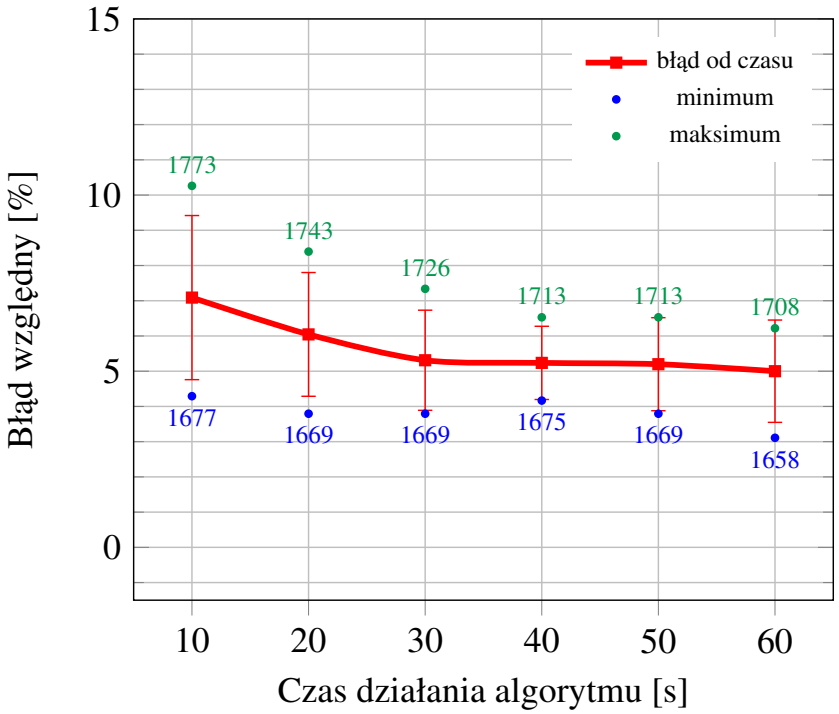
Rysunek 7: Wyniki działania algorytmu *Tabu search* dla instancji *p43.atsp*



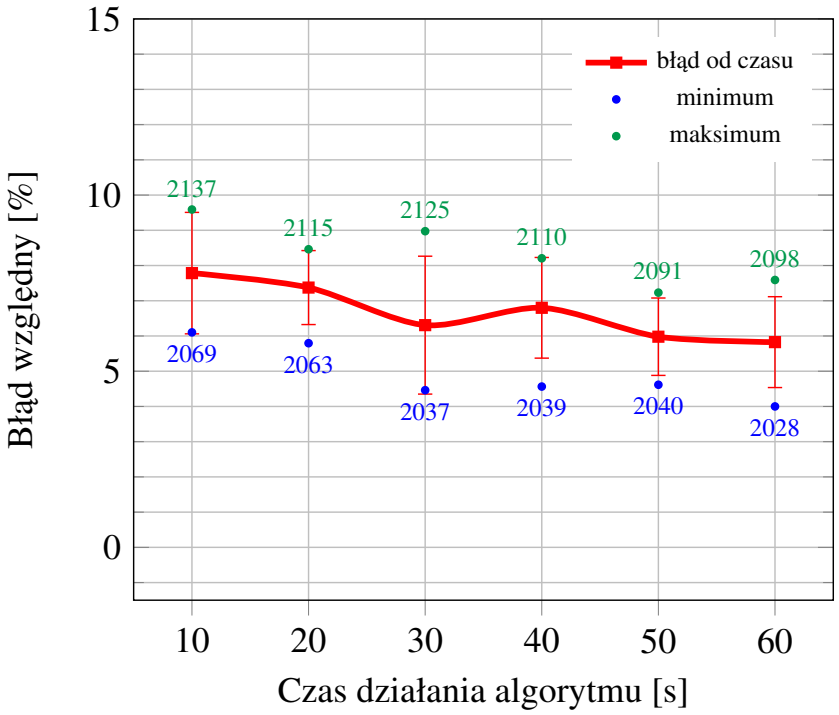
Rysunek 8: Wyniki działania algorytmu *Tabu search* dla instancji *ry48p.atsp*



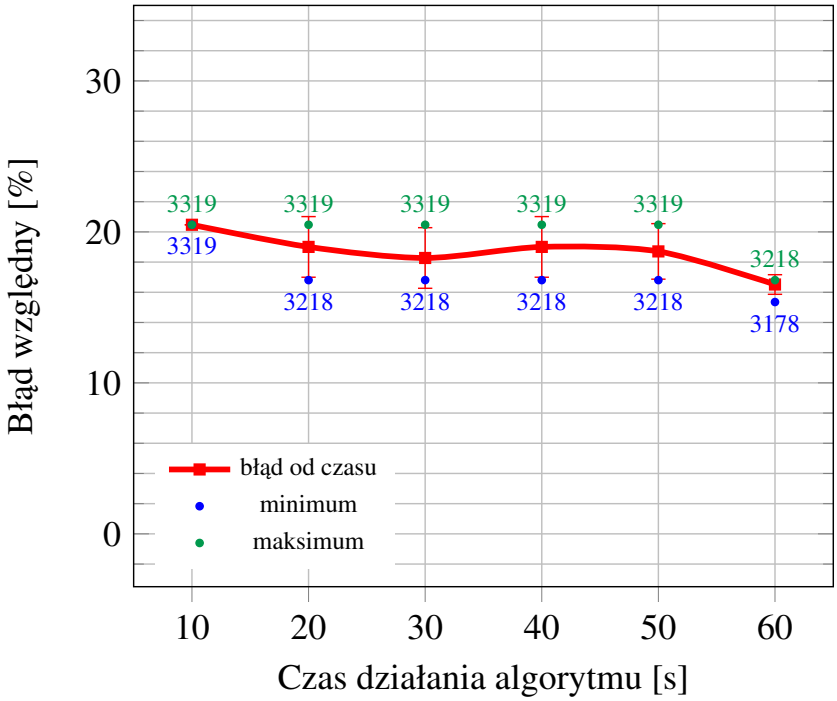
Rysunek 9: Wyniki działania algorytmu *Tabu search* dla instancji *ft53.atsp*



Rysunek 10: Wyniki działania algorytmu *Tabu search* dla instancji *ftv55.atsp*



Rysunek 11: Wyniki działania algorytmu *Tabu search* dla instancji *ftv70.atsp*



Rysunek 12: Wyniki działania algorytmu *Tabu search* dla instancji *ftv170.atsp*

2.2 Symulowane wyżarzanie

Algorytm *Simulated Anealing* (symulowane wyżarzanie) to kolejna z metod przeszukiwania lokalnego - która podobnie jak *Tabu search* - bazuje na dynamicznej zmianie sąsiedztwa danego rozwiązania, ale w odróżnieniu od *Tabu* nie przeszukuje całego sąsiedztwa i zmiana zachodzi pod pewnym, ściśle określonym matematycznym równaniem prawdopodobieństwem. Algorytm powstał na podstawie algorytmu autorstwa N. Metropolisa, służącemu do symulacji zachowań grupy atomów znajdujących się w równowadze termodynamicznej przy zadanej temperaturze. Zamiast zmiany energii zostały wprowadzone pojęcia nowej i starej wartości funkcji celu, zaś początkowa temperatura w algorytmie zastępuje początkową energię.

Bardzo ogólnie - algorytm polega na losowym przetasowaniu ścieżki i przyjęciu nowego rozwiązania jeżeli jest lepsze, a jeżeli jest gorsze to przyjęcie go z pewnym prawdopodobieństwem. Umożliwia to wychodzenie poza obszar minimum lokalnego, co znacznie ułatwia odnalezienie minimum globalnego. Niezbędne jest jednak odpowiednie „nastrojenie" algorytmu, czyli ustawienie specyficznych dla niego parametrów, które i tak nie daje pewności znalezienia minimum globalnego, gdyż zawsze występuje pewien czynnik losowy - w generowaniu sąsiada i w funkcji prawdopodobieństwa.

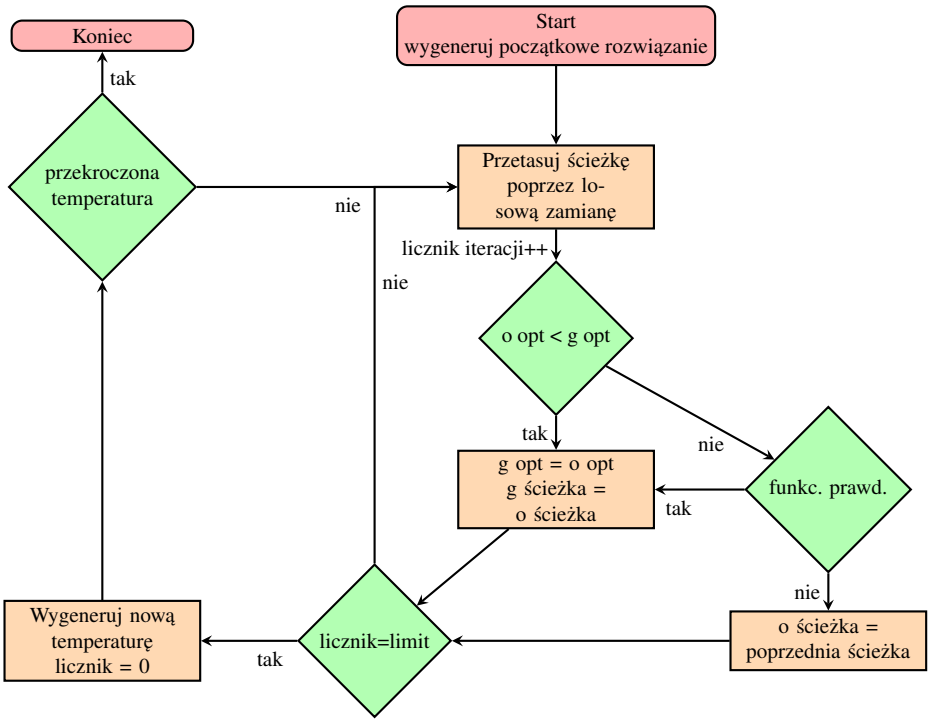
2.2.1 Parametry zaimplementowanego Symulowanego wyżarzania

- **początkowa temperatura** - temperatura na starcie programu,
- **minimalna temperatura** - temperatura, której przekroczenie skutkuje zakończeniem działania algorytmu,
- **funkcja prawdopodobieństwa** - funkcja decydująca o zaakceptowaniu sąsiada,

- **długość epoki** - liczba wewnętrznych iteracji dla jednej temperatury,
- **funkcja wychładzania** - funkcja zmniejszająca temperaturę,
- **globalnie najlepsza ścieżka** - ścieżka, która dotychczasowo dała najmniejszą funkcję celu,
- **obecna ścieżka** - obecnie eksplorowana ścieżka,
- **globalne optimum** - funkcja celu globalnie najlepszej ścieżki,

2.2.2 Opis i schemat działania algorytmu

Działanie algorytmu rozpoczyna się od wygenerowania wybranym algorytmem początkowej ścieżki. Licznik wewnętrznych iteracji jest równy 0 i dokonywana jest pierwsza, losowa zamiana ścieżki wybranym rodzaj sąsiedztwa - *reverse*, *insert* albo *swap* i inkrementowany jest licznik. Jeżeli funkcja celu **obecnej ścieżki** jest mniejsza niż **globalne optimum** to obecna ścieżka staje się globalnie najlepszą ścieżką i analogicznie funkcja celu staje się globalnym optimum.Jeżeli nie jest równa lub większa to jest zgodnie z pewnym prawdodobieństwem, wyliczonym za pomocą **funkcji prawdopodobieństwa** - uproszczonego algorytmu Metropolisa (w fizyce „*prawdopodobieństwo zmiany poziomu energii układu maleje wraz ze wzrostem energii cząsteczki o* δE *oraz ze spadkiem temperatury.*"). Występuje tutaj wspomniana wcześniej spora analogia - zmiana poziomu energii układu to nic innego jak zmiana wartości globalnego rozwiązania, a do tych zmiana dochodzi coraz rzadziej jeżeli temperatura jest niższa. Jeżeli funkcja nie pozwoliła na zmianę to następuje powrót do poprzedniej ścieżki i sprawdzane jest czy liczba wewnętrznych iteracji nie przekroczyła **długości epoki**. Jeżeli nie to ponownie następuje przetasowanie obecnej ścieżki poprzez zamianę zgodnie z rodzajem sąsiedztwa, zaś jeśli tak to generowana (poprzez **funkcję wychładzania**) jest nowa temperatura za pomocą funkcji chłodzenia, która polega na przemnożeniu obecnej temperatury przez stały współczynnik α z zakresu $< 0.85, 0.99 >$. Sprawdzane jest czy nowa temperatura nie przekroczyła **minimalnej temperatury** - jeżeli nie to następuje przetasowanie, a jeżeli tak to algorytm kończy swoją pracę.



Rysunek 13: Schemat *Symulowanego wyżarzania*

2.2.3 Testy

Instancja	Alias	Początkowa temperatura	Minimalna temperatura	Długość epoki	Współczynnik chłodzenia	Algorytm	Sąsiedztwo	Rozwiązanie	Znalezione minimum	Błąd względny
gr17	data17	*100*	*0.1*	*10*	*0.90*	*true*	*0*	2085	2085	0,00%
gr21	data21	*100*	*0.1*	*10*	*0.99*	*false*	*1*	2707	2795	3,25%
gr24	data24	*100*	*0.1*	*100*	*0.85*	*true*	*2*	1272	1379	8,41%
fri26	data26	*100*	*0.1*	*100*	*0.85*	*false*	*2*	937	937	0,00%
bayg29	data29	*100*	*0.1*	*100*	*0.99*	*false*	*0*	1610	1689	4,91%
dantzig42	data42	*100*	*0.1*	*100*	*0.99*	*false*	*2*	699	704	0,72%
brazil58	data58	*100*	*0.1*	*100*	*0.99*	*true*	*1*	25395	27534	8,42%
gr120	data120	*100*	*0.1*	*10*	*0.85*	*false*	*1*	6942	7356	5,96%
swiss42	swiss42	*100*	*0.1*	*100*	*0.95*	*false*	*2*	1273	1273	0,00%

Tablica 3: Tabela najlepszych ustawień parametrów *Tabu* dla danych instancji symetrycznych (TSP) wraz z najlepszym znalezionym rozwiązaniem

Instancja	Alias	Początkowa temperatura	Minimalna temperatura	Długość epoki	Współczynnik chłodzenia	Algorytm	Sąsiedztwo	Rozwiązanie	Znalezione minimum	Błąd względny
br17	data17v2	*100*	*0.01*	*1000*	*0.95*	*true*	*0*	39	39	0,00%
ftv33	data34	*1000*	*0.001*	*1000*	*0.85*	*false*	*1*	1286	1302	1,24%
ftv35	data36	*1000*	*0.01*	*1000*	*0.99*	*false*	*1*	1473	1548	5,09%
ftv38	data39	*100*	*0.001*	*1000*	*0.98*	*false*	*1*	1530	1536	0,39%
p43	data43	*100*	*0.01*	*1000*	*0.85*	*false*	*2*	5620	5620	0,00%
ftv44	data45	*100*	*0.1*	*1000*	*0.90*	*false*	*1*	1613	1653	2,48%
ry48p	data48	*1000*	*0.01*	*1000*	*0.95*	*false*	*1*	14422	14429	0,05%
ftv53	data53	*100*	*0.001*	*1000*	*0.90*	*false*	*1*	6905	7119	3,10%
ftv55	data56	*1000*	*0.001*	*1000*	*0.97*	*false*	*1*	1608	1632	1,49%
ftv64	data65	*100*	*0.1*	*1000*	*0.99*	*false*	*1*	1839	1905	3,59%
ft70	data70	*1000*	*0.1*	*1000*	*0.95*	*false*	*1*	38673	39762	2,82%
ftv70	data71	*1000*	*0.001*	*1000*	*0.99*	*true*	*1*	1950	2113	8,36%
kro124	data100	*1000*	*0.1*	*1000*	*0.99*	*false*	*2*	36230	38448	6,12%
ftv170	data171	*100*	*0.001*	*1000*	*0.99*	*true*	*1*	2755	3340	21,23%
rbg323	data323	*1000*	*0.001*	*1000*	*0.99*	*true*	*0*	1326	1516	14,33%
rbg358	data358	*1000*	*0.001*	*1000*	*0.99*	*true*	*0*	1163	1398	20,21%
rbg403	data403	*1000*	*0.001*	*1000*	*0.99*	*true*	*1*	2465	2544	3,20%
rbg443	data443	*1000*	*0.001*	*1000*	*0.99*	*true*	*1*	2720	2828	3,97%
ftv47	ftv47	*1000*	*0.1*	*1000*	*0.99*	*true*	*1*	1776	1843	3,77%

Tablica 4: Tabela najlepszych ustawień parametrów *Tabu* dla danych instancji asymetrycznych (ATSP) wraz z najlepszym znalezionym rozwiązaniem

2.2.3.1 Pierwsza faza testów

W tablicach 3 i 4 zostały zaprezentowane ustawienia parametrów dla danych instancji, które pozwoliły uzyskać możliwie najbliższe rozwiązaniu wyniki. Pierwsza faza testów polegała na testowaniu różnych ustawień parametrów *wyżarzania* za pomocą zagnieżdżonych pętli. Dla każdej kombinacji parametrów zostało przeprowadzonych po 50 pomiarów.

Wartości *true* i *false* w parametrze *Algorytm* oznaczają odpowiednio skorzystanie z zachłannego i redukcyjnego algorytmu na początku działania algorytmu. Zaś jeśli chodzi o parametr *Sąsiedztwo* to wartości 0,1 i 2 - podobnie jak w *Tabu* - oznaczają odpowiednio uzyskiwanie sąsiedztwa po zamianach typu *reverse*, *swap* oraz *insert*.

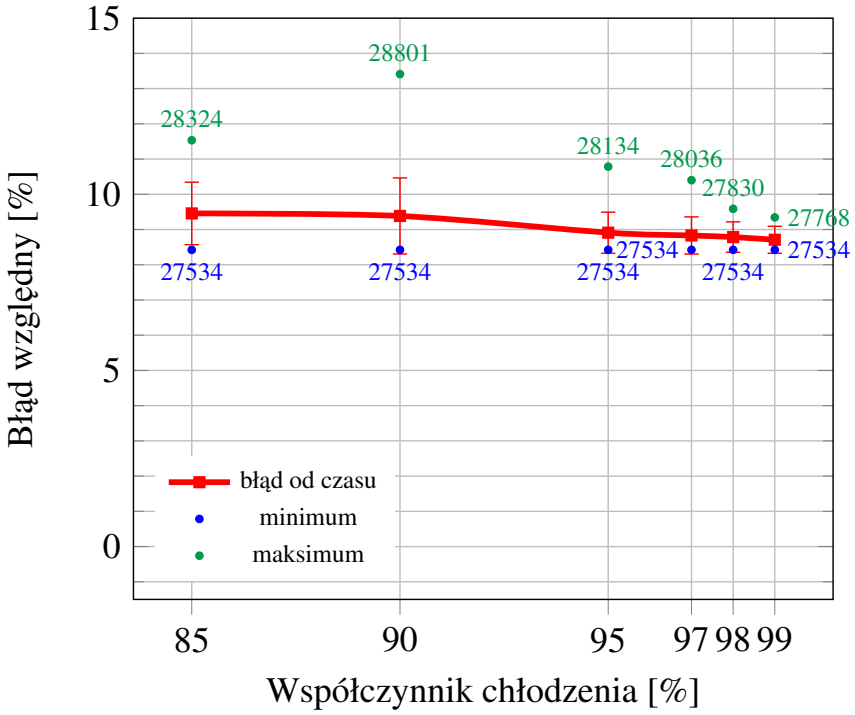
2.2.3.2 Druga faza testów

Druga faza testów polegała na wielokrotnym (100 iteracji) wykonaniu algorytmu dla znalezionej wcześniej, najlepszej kombinacji parametrów dla danej instancji. Najmniejsze-najlepsze rozwiązanie zostało wpisane w kolumnę *Najlepszy rezultat* w tablicach 3 i 4.

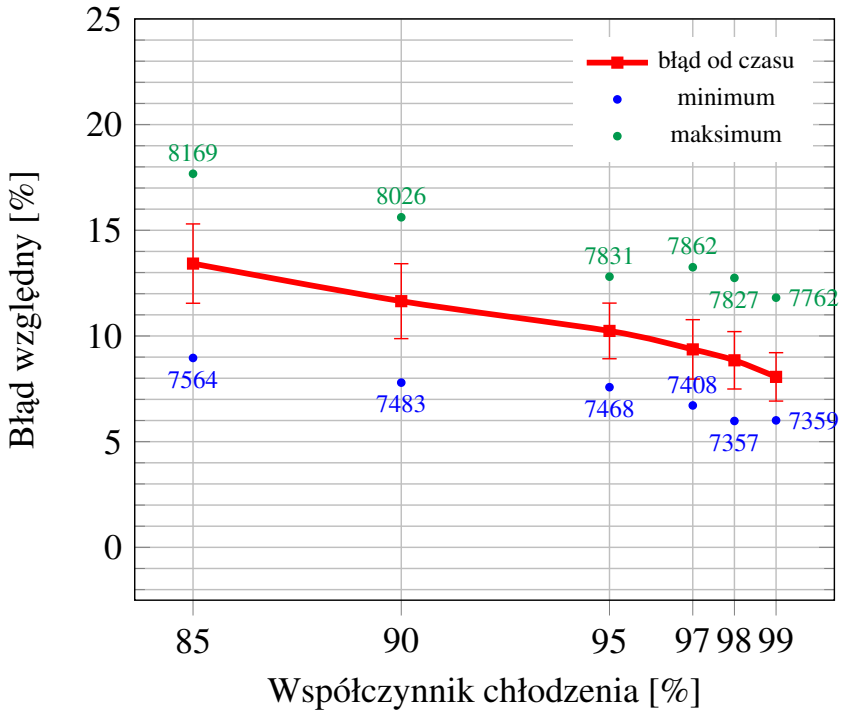
2.2.3.3 Trzecia faza testów

W ramach trzeciej fazy testów zostało wybranych - znów podobnie jak w *Tabu* - 11 instancji - *brazil58*, *gr120*, *ftv33*, *ftv35*, *ftv38*, *p43*, *ry48p*, *ftv53*, *ftv55*, *ftv70* oraz *ftv170* - na których zostały przeprowadzone testy, których celem było zobrazowanie zależności średniego błędu względnego od współczynnika chłodzenia temperatury. Dla wszystkich instancji zostało wybranych 6 współczynników - 0.85, 0.90, 0.95, 0.97, 0.98 oraz 0.99, dla których - wraz z optymalnymi parametrami dla danej instancji - zostało przeprowadzone po 50 testów, a średnie wyniki - wraz z odchyleniami standardowymi - zostały ukazane na poniższych wykresach. Zielone kropki na wykresach oznaczają maksymalne (najgorsze) rozwiązanie znalezione dla danego czasu, a niebieskie minimalne (najlepsze).

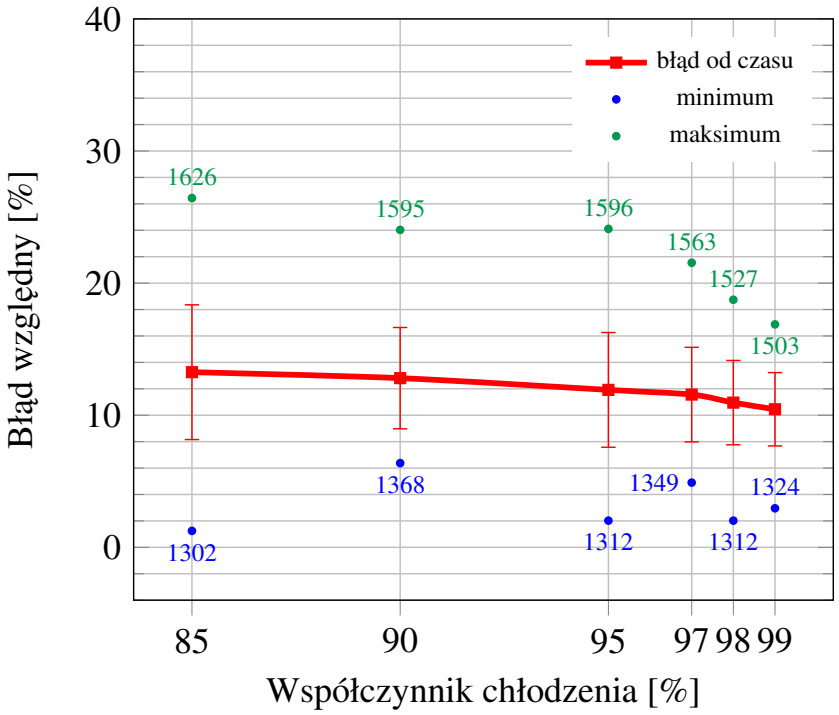
2.2.4 Wyniki testów



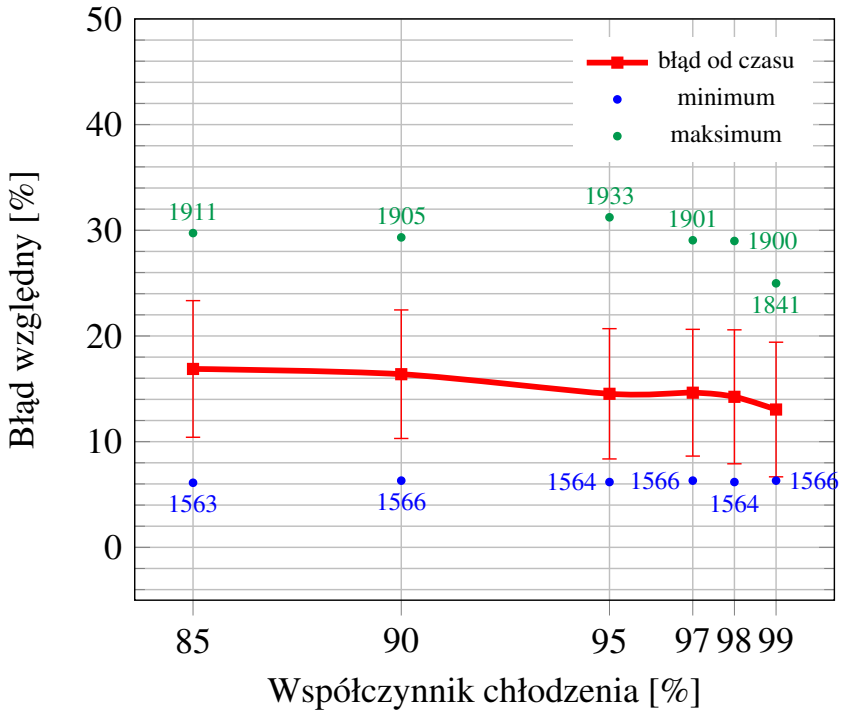
Rysunek 14: Wyniki działania algorytmu *Symulowane wyżarzanie* dla instancji *brazil58.tsp*



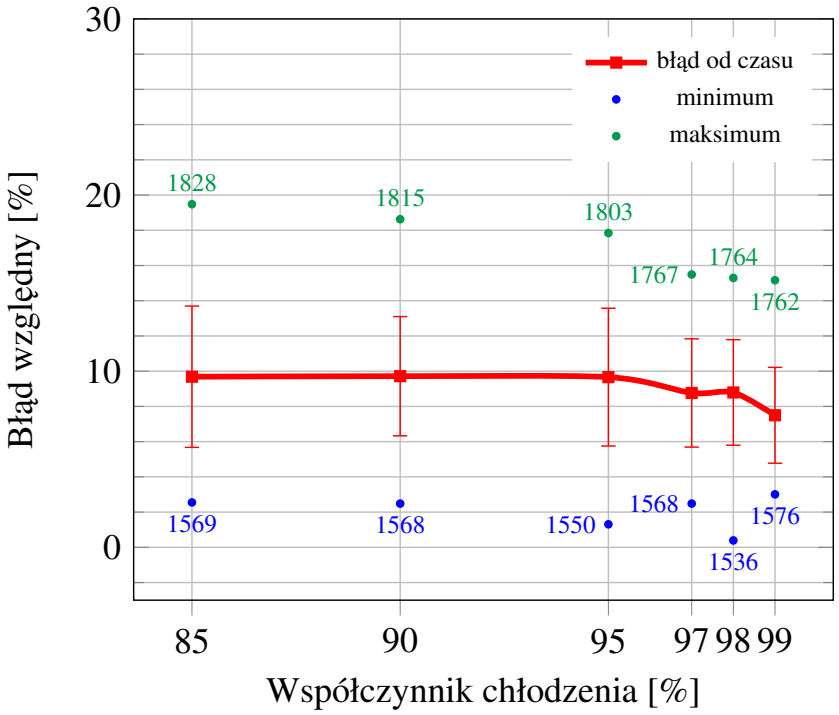
Rysunek 15: Wyniki działania algorytmu *Symulowane wyżarzanie* dla instancji *gr120.tsp*



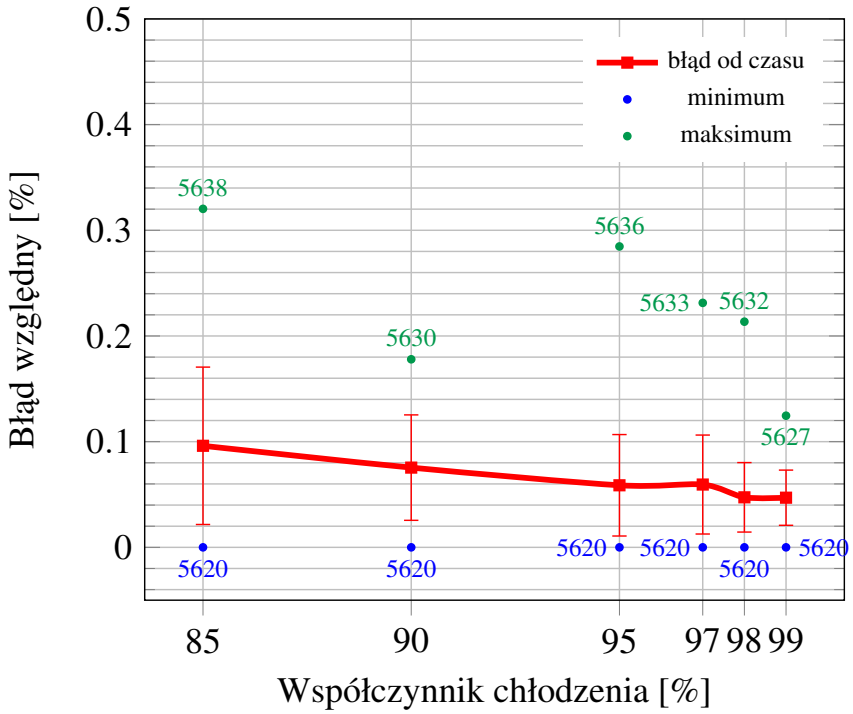
Rysunek 16: Wyniki działania algorytmu *Symulowane wyżarzanie* dla instancji *ftv33.atsp*



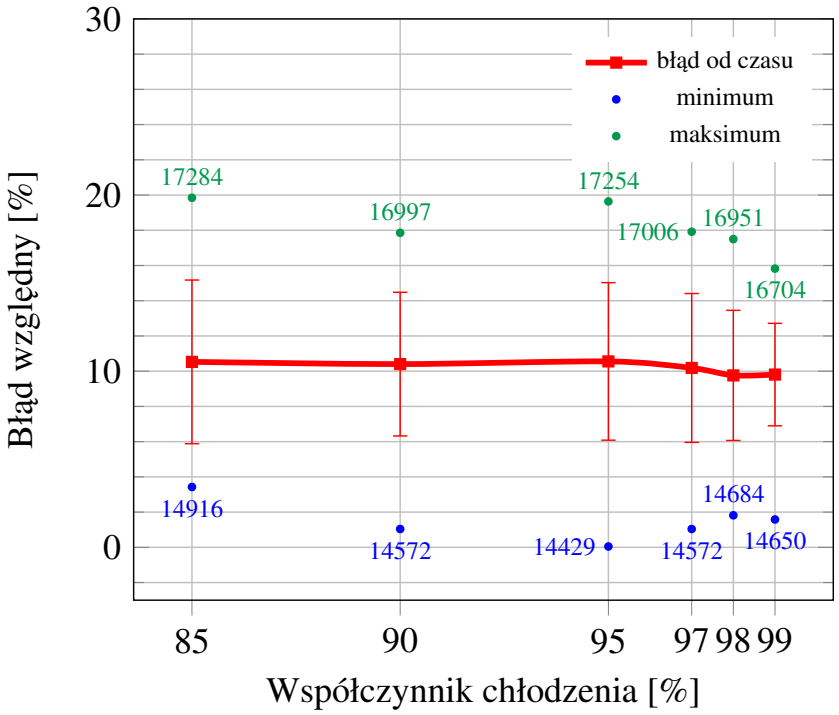
Rysunek 17: Wyniki działania algorytmu *Symulowane wyżarzanie* dla instancji *ftv35.atsp*



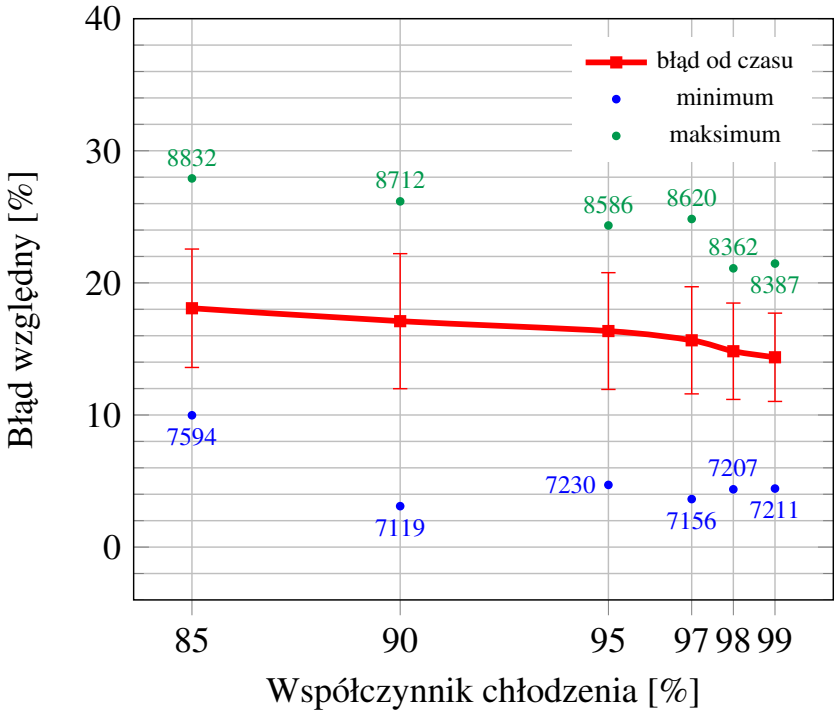
Rysunek 18: Wyniki działania algorytmu *Symulowane wyżarzanie* dla instancji *ftv38.atsp*



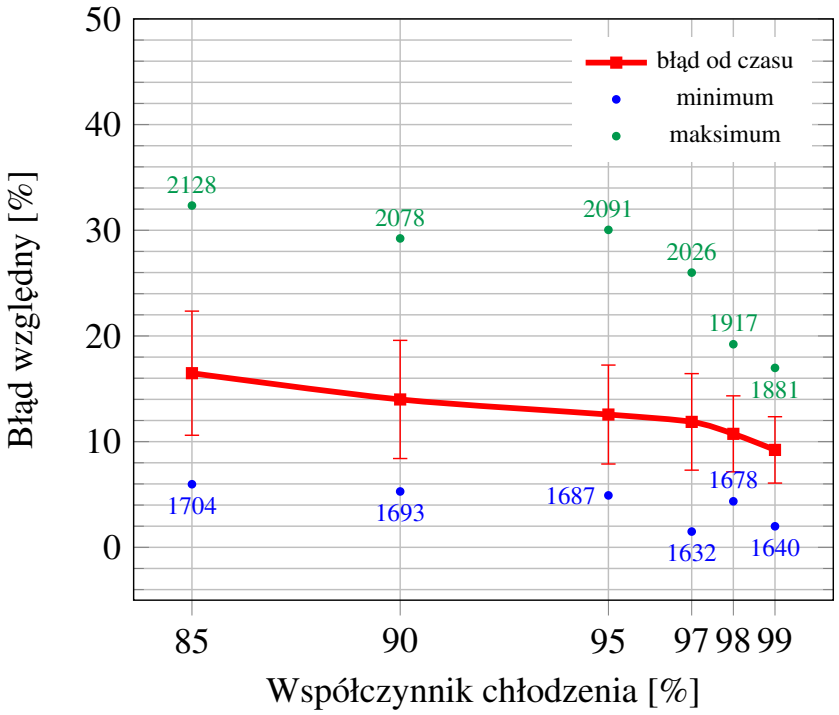
Rysunek 19: Wyniki działania algorytmu *Symulowane wyżarzanie* dla instancji *p43.atsp*



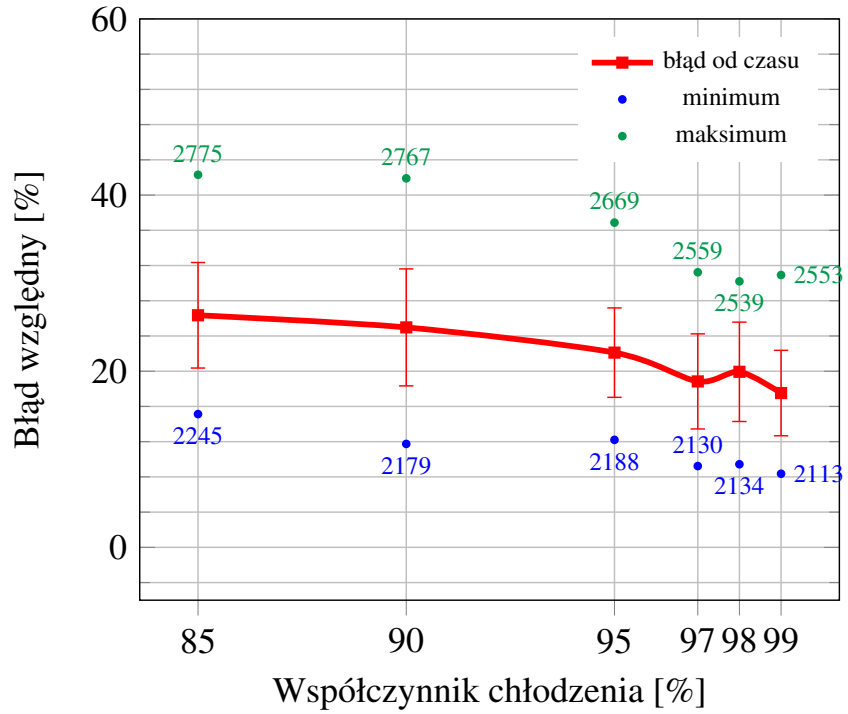
Rysunek 20: Wyniki działania algorytmu *Symulowane wyżarzanie* dla instancji *ry48p.atsp*



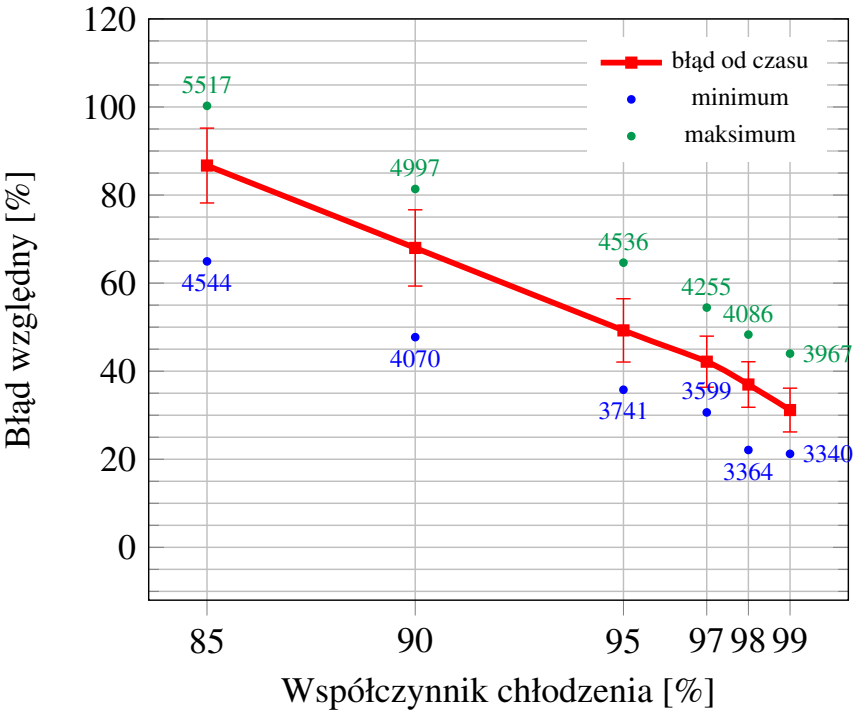
Rysunek 21: Wyniki działania algorytmu *Symulowane wyżarzanie* dla instancji *ftv53.atsp*



Rysunek 22: Wyniki działania algorytmu *Symulowane wyżarzanie* dla instancji *ftv55.atsp*



Rysunek 23: Wyniki działania algorytmu *Symulowane wyżarzanie* dla instancji *ftv70.atsp*



Rysunek 24: Wyniki działania algorytmu *Symulowane wyżarzanie* dla instancji *ftv170.atsp*

3 Wnioski

Zaprezentowane metody metaheurystyczne, dostosowane konkretnie pod problem komiwojażera mogą być bardzo efektywne pod warunkiem dobrej, przemyślanej implementacji i odpowiedniego „nastrojenia" algorytmu pod konkretną instancję. Projekt można było podzielić na dwie części - implementację oraz testowanie, z których to drugie zajęło zdecydowanie więcej czasu, co już jest pierwszą rzeczą, która odróżnia algorytmy dokładne z wcześniejszego etapu od zaprezentowanych tutaj metaheurystycznych metod lokalnego przeszukiwania. Uzyskiwane rezultaty można poprawić poprzez wykorzystanie chociażby metod z poprzedniego projektu z wspomnianych algorytmów dokładnych.

W przypadku implementacji *Tabu search* został wykonany hybrydowy algorytm działający tak samo jak początkowy algorytm z *Branch&Bound*, lecz przyjmujący na początku - przekazaną jako parametr - ilość losowych wierzchołków na początku ścieżki, co zapewniło już na początku wynik i ścieżkę znacznie bliższą optymalnym niż byłoby to po wygenerowaniu ścieżki całkowicie losowo. Ważnym aspektem implementacji było skorzystanie z wielu rodzajów sąsiedztwa ścieżek. Przykładowo skorzystanie z sąsiedztwa uzyskiwanego po ruchu *reverse* pozwalało na uzyskanie optymalnego rozwiązania w problemach TSP w maksymalnie jedną sekundę w 6 z 9 problemów, a w maksymalnie pięć sekund w 8 z 9. Analizując tablice 1-4 widać, że każde z sąsiedztw okazało się być lepsze od innych w przynajmniej paru instancjach. Implementując różne rodzaje sąsiedztwa warto zastanowić się jak zapisać ruch *insert*, gdyż przykładowo zakazując ruchu *insert(6,1)* (przesunięcie wierzchołka 6 na pozycję nr 1) zdecydowanie bardziej korzystne okazuje się przeznaczyć jedną kolumnę listy Tabu nie na pozycję, na którą był przesuwany wierzchołek, lecz np. na wierzchołek za którym bądź przed którym się teraz znajduje wierzchołek nr 6. Taki sposób zapisu okazał się być - po szybkim przetestowaniu - bardzo korzystny i w problemach ATSP aż 11 na 19 razy okazał się być najlepszy, głównie dla problemów o większym *n*. Jeszcze jednym ważnym aspektem generowania sąsiadów było *szybsze liczenie sąsiedniego ruchu*, czyli konkretnie różnicy między poprzednią i nową funkcją celu. W przypadku każdego z trzech ruch łatwo można doszukać się pewnych prawidłowości i po analizie przetasowania ścieżki zapisać, które krawędzie są dodawane, a które odejmowane, co zostało opisane w punkcie 2.1.4. Rozwiązanie takie prezentuje się znacznie efektywniej, zwłaszcza dla problemów o dużym *n*, gdyż pozwala uniknąć dodawania przykładowo 400 wartości, a zamiast tego - konkretnie dla ruchu *swap* - wystarczy dodać i usunąć po 4 krawędzie.

Jeżeli chodzi o implementację *symulowanego wyżarzania* to zostały tutaj wykorzystane metody każdego z rodzajów sąsiedztwa z *Tabu*. Również podobnie jak w *Tabu* na początku ścieżka była generowana *branch&boundowym* algorytmem. Kluczowym okazało się tutaj właściwe dobranie parametrów. Funkcja prawdopodobieństwa została skonstruowana na podstawie algorytmu Metropolisa, a funkcja chłodzaca temperaturę przyjęła bardzo prostą wersję - mnożenia przez stały współczynnik bliski 1. Zostało przetestowanych ponadto parę innych funkcji chłodzących, trochę bardziej skomplikowanych [1], lecz po wstępnych testach nie dawały ulepszeń, a wręcz przeciwnie, wobec czego została zachowana jej pierwotna wersja.

Ciężko właściwie porównać oba algorytmy, gdyż można zdefiniować różne warunki zakończenia działania algorytmu, dla których jeden może się prezentować gorzej, a drugi lepiej i na odwrót. Patrząc na wykresy obu algorytmów widać, że dla coraz większych argumentów z osi OX (czas lub współczynnik chłodzenia) maleją maksima uzyskiwane w ramach danej liczby pomiarów dla instancji, lecz jeśli chodzi o minima to nie da się jednoznacznie określić co właściwie dzieje się z uzyskiwanymi rezultatami, szczególnie dla *symulowanego wyżarzania*. Patrząć na tablice 3 i 4 i kolumnę współczynnik chłodzenia widać - szczególnie dla mniejszych instancji - że minima zostały znalezione dla mniejszych współczynników. Jest to spowodowane tym, że temperatura szybciej maleje i zaimplementowana funkcja prawdopodobieństwa, która bazuje na temperaturze, nie akceptuje - mogących dać w dalszych iteracjach korzyści - ścieżek. Z *Tabu* jest podobnie, lecz tutaj chodzi wyłącznie o czynnik losowy, gdyż nowa ścieżka generowana jest z losowymi wierzchołkami na początku i przez stosunkowo małą liczbę pomiarów (po 5 dla jednej kombinacji parametrów) na wykresach widać, że dla niektórych instancji mniejsze czasy dały lepsze rezultaty, choć *Tabu* - w odróżnieniu od *symulowanego wyżarzania* nie akceptuje gorszych rozwiązań ze względu na specyfikę zaimplementowanego kryterium aspiracji. Jednakże na wykresach da się zauważyć wyraźną tendecję do zmniejszania się średniego błędu względnego oraz jego odchylenia standardowego.

Podsumowując, z korzystania z metod lokalnego przeszukiwania płyną korzyści szczególnie wtedy gdy nie zależy nam na uzyskaniu optymalnego wyniku, lecz jedynie pewnego przybliżenia. Znacznie prostsze jest ustawienie parametrów *symulowanego wyżarzania* niż *Tabu*, lecz właściwie nastrojony *Tabu* spisuje się naprawdę dobrze - dla instancji

o $n < 100$ dla kilkudziesięciu pomiarów, najlepszy wynik był z błędem maksymalnie 3.11% (w większości mniejszy od 1%). Możliwe jest uzyskanie przybliżone rozwiązania dla problemów znacznie przerastających możliwości algorytmów dokładnych, tj. dla n równych już chociażby 40.

Bibliografia

[1] A Comparison of Cooling Schedules for Simulated Annealing (Artificial Intelligence). http://what-when-how.com/artificial-intelligence/a-comparison-of-cooling-schedules-for-simulated-annealing-artificial-intelligence/.

[2] Seminarium: Algorytmy heurystyczne. https://www.ii.uni.wroc.pl/~prz/2011lato/ah/opracowania/szuk_tabu.opr.pdf.

[3] Tabu Search Method for Solving the Traveling salesman Problem. https://www.iasj.net/iasj?func=fulltext&aId=7908.

[4] Robert Kruse. *Data structures and program design*. Prentice-Hall, 1984.

[5] Radosław Lis. Listing kodu. https://github.com/radosz99/PEA. [Online; accessed 17-December-2019].

[6] Stanisław Mikulski. WYKORZYSTANIE METODY SYMULOWANEGO WYŻARZANIA DO OPTYMALIZACJI UŁAMKOWEGO REGULATORA. pages 106–109, 2015.

[7] Krzysztof Olszowy. Aplikacja znajdująca najkrótszą drogę w supermarkecie. Master's thesis, Akademia Górniczo-Hutnicza w Krakowie.