IMPLEMENTACJA I ANALIZA EFEKTYWNOŚCI ALGORYTMU TABU SEARCH I SYMULOWANEGO WYŻARZANIA DLA WYBRANEGO PROBLEMU OPTYMALIZACJI

Projektowanie efektywnych algorytmów

AUTOR: MACIEJ BRONIKOWSKI 248838
GRUPA PROJEKTOWA: ŚRODA 13:15
PROWADZĄCY: ANTONI STERNA

1 WSTĘP TEORETYCZNY

Przedmiotem projektu było zaprojektowanie algorytmów metaheurystycznych Tabu Search oraz Symulowanego Wyżarzania dla jednoprocesorowego problemu szeregowania zadań przy kryterium minimalizacji ważonej sumy opóźnień zadań. Algorytmy te używane są do znajdowania przybliżenia globalnej optymalizacji problemów, dla których przeszukanie pełnej przestrzeni możliwych rozwiązań jest zbyt czasochłonne.

Tabu Search oraz Symulowane wyżarzanie są pewną wariacją algorytmu przeszukiwania lokalnego, którego algorytm można przedstawić w następujący sposób:

x_0 <- random(X)
do
  x <- N(x_0)
  if f(x) < f(x_0) **then**
    x_0 <- x
  end if
while f(x) < f(x_0) dla wszystkich x ∈ $N(x_0)$

Algorytm 1 Algorytm przeszukiwania lokalnego

Gdzie:

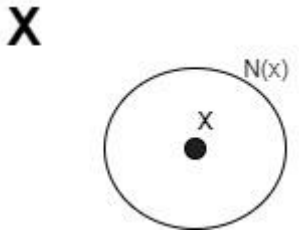
- X – przestrzeń wszystkich możliwych rozwiązań problemu
- X_0 – najlepsze dotąd znalezione rozwiązanie
- N(x) – generowanie sąsiada rozwiązania x
- f(x) – funkcja przedstawiająca wartość rozwiązania, której minimum staramy się znaleźć
- random(X) – generowanie losowego rozwiązania problemu z przestrzenią rozwiązań X

Dla powyższego algorytmu możemy zaimplementować dwie wersje przeszukiwania przestrzeni sąsiadów rozwiązania x_0:

- Greedy – algorytm przechodzi do pierwszego lepszego znalezionego rozwiązania
- Steepest – algorytm przeszukuje całą przestrzeń sąsiadów rozwiązania i wybiera najlepsze

Dodatkowo dla algorytmu przeszukiwania lokalnego należy zdefiniować czym jest sąsiedztwo rozwiązania x.

2 SĄSIEDZTWO



Rysunek 1 Pojęcie sąsiedztwa

Sąsiedztwo N(x) rozwiązania x jest jest pewnym zbiorem rozwiązań, gdzie N(x) ∈ X. Sąsiedztwo powinno posiadać rozwiązania bliskie rozwiązaniu x, oraz być na tyle małe, aby przeszukiwanie wszystkich możliwych sąsiadów x nie było czasochłonne.

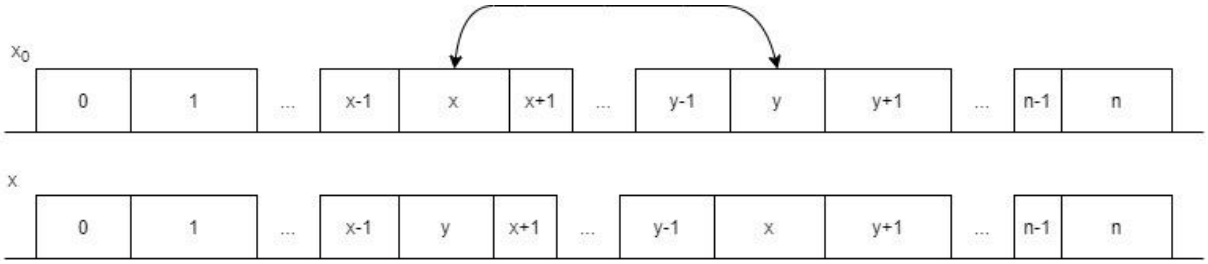
Podczas realizacji projektu zaimplementowane zostały dwa rodzaje sąsiedztwa:

- Exchange
- Insert

2.1 SĄSIEDZTWO TYPU EXCHANGE

2.1.1 Opis sąsiedztwa

Sąsiedztwo to polega na zamianie miedzy sobą dwóch elementów o wybranych indeksach. Dla rozpatrywanego problemu, przy założeniu, że posiada on n zadań, oraz chcemy zamienić między sobą indeksy zadań x oraz y, operacja ta wyglądałaby w następujący sposób:



Rysunek 2 sąsiedztwo typu exchange

Jak widać na powyższym rysunku, zamiana taka powoduje przesunięcie czasu wykonania się zadań x+1, x+2, … , y-2, y-1. Aby obliczyć wartość nowego rozwiązania x, można by użyć następującego wzoru

$$f(x) = f(x_0) + ([\sum_{i=x}^{y}(w_i * T_i)] - [(w_y * T_y + \sum_{i=x+1}^{y-1}(w_i * T_i) + w_x * T_x)])$$

Gdzie:

- w_i – waga zadania
- T_i – wartość opóźnienia zadania
- Wartość opóźnienia w nawiasach [] jest liczona tak, jakby zadania były wykonane jedno po drugim, a pierwsze z nich zostało wykonane po zadaniu x-1.

2.1.2 Rozmiar sąsiedztwa

Obliczanie sumy ważonych opóźnień x zostało zaimplementowane w taki sposób, że cała suma ważonych opóźnień obliczana jest na nowo z uwagi, że złożoność jest podobna do sposobu przedstawionego w wzorze powyżej.

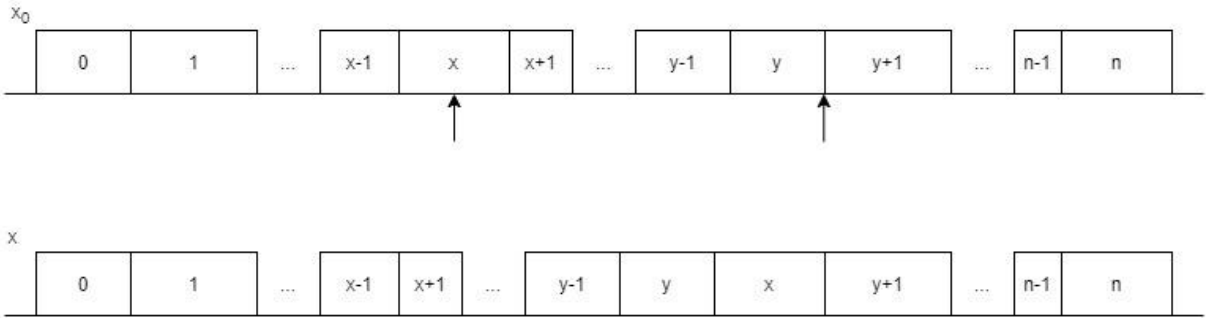
Rozmiar opisanego sąsiedztwa z uwagi że operacje zamiany (x,y) oraz (y,x) są równoważne wynosi

$$|N_{exchange}(x)| = \frac{n(n-1)}{2}$$

2.2 SĄSIEDZTWO TYPU INSERT

2.2.1 Opis sąsiedztwa

Sąsiedztwo to polega na pobraniu elementu z rozwiązania i wstawienie go w inne miejsce. Dla rozpatrywanego problemu, przy założeniu, że posiada on n zadań, oraz chcemy zamienić między sobą indeksy zadań x oraz y, operacja ta wyglądałaby w następujący sposób:



Rysunek 3 Sąsiedztwo typu insert

W projekcie obliczanie wartości f(x) sąsiada zostało zaimplementowane przez obliczanie pełnej sumy ważonego opóźnienia z powodów wymienionych już w punkcie 2.1.

2.2.2 Rozmiar sąsiedztwa

Rozmiar opisanego sąsiedztwa różni się od rozmiaru sąsiedztwa typu exchange. Pobranie elementu x i wstawienie go za element y wygeneruje inne rozwiązanie niż pobranie elementu y i wstawienie go za element x. Wartość tego rozmiaru można przedstawić za pomocą wzoru:

$$|N_{insert}(x)| = n^2$$

3 ALGORYTMY UŻYTE W PROJEKCIE

3.1 ALGORYTM SYMULOWANIA WYŻARZANIA

Działanie algorytmu przypomina zjawisko wyżarzania w metalurgii. Wyżarzanie w metalurgii polega na podgrzaniu pewnego ciała do wartości, w której zaczyna topnieć, a następnie na polnym zmniejszaniu jego temperatury do chwili, aż cząsteczki materiału osiągną przybliżony stan równowagi.

Algorytm symulowanego wyżarzania można przedstawić w następujący sposób:

x_0 <- random(X)
c <- C_0
k <- 0
x_b <- f(x_0)
do
  for i **in** 0 … L_k **do**
    x <- N(x_0)
    if f(x) < f(x_o) **then**
      x_0 <- x
      if f(x_b) > f(x_0) **then**
        x_b <- x_0
      end if
    else if exp(- (f(x) - f(x_o)) / c) < random[0,1) **then**
      x_0 <- x
    end if
  end for
  k <- k + 1
  c <- C(k)
while warunek_stopu

Algorytm 2 Symulowanie Wyżarzania

Gdzie:

- c – obecnie panująca temperatura
- C_0 – temperatura początkowa
- k – numer iteracji algorytmu
- L_k – ilość iteracji, przez które w obecnie panującej temperaturze będziemy próbować osiągnąć stan równowagi
- C(k) – schemat chłodzenia. Funkcja obliczająca kolejną temperaturę dla iteracji k.
- warunek_stopu – warunek, dla którego algorytm zostanie zatrzymany. Warunkiem tym może być przykładowo schłodzenie do wyznaczonej temperatury, lub osiągnięcie wyznaczonego czasu działania algorytmu
- x_b – z uwagi na to, że przy odpowiednio wysokiej temperaturze najlepsze znalezione dotąd rozwiązanie może zostać zastąpione mniej optymalnym, należy przechować dodatkową informację na temat najlepszego znalezionego rozwiązania

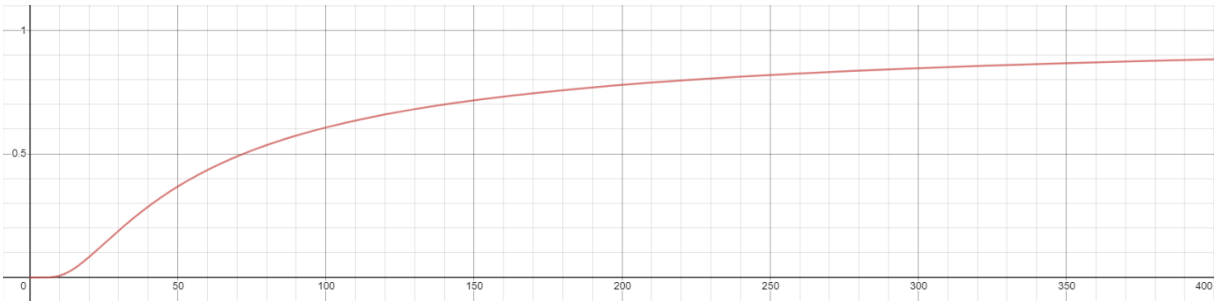
Jak zostało opisane w punkcie 1, Symulowane wyżarzanie przypomina swoją budową algorytm lokalnego przeszukiwania, a dokładniej jego odmianę Greedy.

3.1.1 Kryterium akceptacji

Kryterium akceptacji jest pewnym prawdopodobieństwem, dla którego sąsiad x zostanie zaakceptowany jako x_0 do którego będzie się odnosić następna iteracja pętli. Prawdopodobieństwo to można zapisać za pomocą wzoru

$$P_c\{accept\ x\} = \begin{cases} 1, & f(x) \leq f(x_0) \\ e^{\frac{f(x_0)-f(x)}{c}}, & f(x) > f(x_0) \end{cases}$$

Jak widać na powyższym wzorze, jeżeli wartość sąsiada jest mniejsza od wartości obecnie przetwarzanego rozwiązania, sąsiad zawsze zostanie zaakceptowany jako nowe przetwarzane rozwiązanie. Jeżeli jednak sąsiednie rozwiązanie nie przynosi zysku, zostanie ono zaakceptowane z pewnym prawdopodobieństwem. Przykładowe wartości funkcji tego prawdopodobieństwa dla pogorszenia równego 100 oraz parametru c zostały zamieszczone na poniższym wykresie:



Jak można zauważyć na wykresie, dla temperatury równej 400, prawdopodobieństwo zaakceptowania rozwiązania pogarszający wynik jest bliskie 1, jednak gdy temperatura spada, prawdopodobieństwo to dąży do zera.

3.1.2 Schemat chłodzenia

Schematem chłodzenia C(k) to pewien algorytm według którego temperatura c będzie maleć podczas wykonywania się kolejnych iteracji algorytmu. Można wyróżnić min:

- Schemat logarytmiczny (Boltzmanna): $C(k) = \frac{C_0}{a+b*\log(k)}$
- Schemat liniowy (Cauchy'ego): $C(k) = \frac{C_0}{a+b*k}$
- Schemat geometryczny: $C(k) = a^k * C_0,\ gdzie\ 0 < a < 1$

Dobór schematu chłodzenia jest ważny, gdyż to on będzie decydować jak będzie zmieniać się prawdopodobieństwo akceptacji pogarszającego wyniku wraz z czasem działania algorytmu.

Dodatkowo w przedstawionym algorytmie Symulowanego Wyżarzania został przedstawiony schemat chłodzenia, gdzie przez pewien okres L_k temperatura pozostaje stała. W zaimplementowanym algorytmie długość ta jest proporcjonalna do rozmiaru sąsiedztwa i dla obu przypadków sąsiedztwa wynosi n^2.

3.2 TABU SEARCH

Tabu Search (z polskiego Przeszukiwanie z zakazami) polega na pomijaniu tych ruchów, które prowadziłyby do powrotu w miejsca już odwiedzone. Algorytm ten można przedstawić w następujący sposób:

x_0 <- random(X)
x_b <- f(x_0)
do
  x_{tmp} <- null
  for all x ∈ N(x_0) **do**
    if (f(x) < f_b **and** x ∉ tabu) **or** f(x) < f(x_b) **then**
      x_{tmp} = x
    end if
  end for
  refresh(tabu, x)
  x_0 <- x
  if f(x_0) < f(x_b) **then**
    x_b <- x_0
  end if
while warunek_stopu

Algorytm 3 Tabu Search

Gdy:

- x_0 – obecnie przetwarzana kolejność zadań
- x_b – najlepsza dotąd znaleziona kolejność zadań (z uwagi na to, że poruszając się po sąsiedztwie można opuścić najlepszy znaleziony wynik, należy go zapamiętać)
- x_{tmp} – przechowywanie najlepszego sąsiada spełniającego kryteria
- tabu – lista ruchów zakazanych
- refresh(tabu, x) – funkcja dodająca ruch odwrotny do wykonanego do listy tabu, oraz odświerzająca i zmniejszająca kadencje ruchów zakazanych
- warunek_stopu – warunek zakończenia działania algorytmu. Może to być przykładowo czas działania algorytmu lub liczba iteracji.

3.2.1 Lista tabu

Lista tabu pozwala na zapobieganie powrotu algorytmu w odwiedzone już rozwiązania. Po wybraniu rozwiązania, które będzie przetwarzane w następnej iteracji należy dodać do listy tabu ilość iteracji algorytmu, przez które odwrotny ruch do wykonanego będzie zakazanych. Informacja ta nazwana jest kadencją.

Kadencje należy zmniejszać podczas każdej iteracji algorytmu. Dodatkowo, jeżeli kadencje są długie, należy zadbać o to, aby algorytm mógł wybrać kolejne rozwiązanie.

3.2.1.1 Lista tabu dla sąsiedztwa typu exchange

Z uwagi na to, że dla sąsiedztwa typu exchange operacje (x,y) oraz (y,x) wyznaczają dokładnie to samo rozwiązanie, ruchem zakazanym jest dokładnie ta sama operacja. Należy jednak pamiętać, aby wybrać jedną z dwóch tożsamych operacji i konsekwentnie jej używać podczas operacji na liście tabu.

Załóżmy, że dla rozmiaru problemu szeregowania zadań równego 5, obecnie przetwarzaną kolejnością zadań jest x_0 = {0,1,2,3,4}. Dla pewnej iteracji przeglądu sąsiedztwa N(x_0) została wykonana operacja zamiany indeksów (1,2), w wyniku której otrzymano rozwiązanie x={0,2,1,3,4}, które okazało się sąsiadem o najlepszym wyniku. Gdyby w kolejnym przebiegu pętli głównej algorytmu okazało się, że najlepszego sąsiada można uzyskać wykonując ponownie operację zamiany indeksów (1,2) otrzymalibyśmy wynik x={0,1,2,3,4} co spowodowałoby powrót do rozwiązania x_0.

3.2.1.2 Lista tabu dla sąsiedztwa typu insert

Dla sąsiedztwa typu insert dla operacji (x,y) należy zakazać ruchu (y,x), z uwagi na to, że rozwiązanie otrzymane w wyniku operacji insert na indeksach (x,y) oraz (y,x) nie są tożsame, jednak wykonane jedna po drugiej powodują powrót do wartości początkowej rozwiązania.

3.2.2 Kryterium aspiracji

Zakazy związane z informacjami przechowywanymi na liście tabu mogą prowadzić do pomijania lepszych rozwiązań. Kryterium aspiracji polega na pomijanie zakazów wynikających z listy tabu, jeżeli mogą one przynieść lepsze rozwiązanie.

W zaimplementowanym algorytmie kryterium aspiracji które zostało zaimplementowane jest używanie zakazanego ruchu, jeżeli prowadzi on do lepszego rozwiązania niż dotychczas znalezione.

3.2.3 Dywersyfikacja

Algorytmy optymalizacji mogą zostać opisane za pomocą umieszczenia ich na osi zamieszonej poniżej:

Pełna intensyfikacja polega na przeszukiwaniu przestrzeni rozwiązań jedynie w miejscu sąsiedztwa. Algorytmem który mógłby zostać umieszony całkowicie po lewej stronie osi jest algorytm przeszukiwania lokalnego.

Pełna dywersyfikacja polega na w pełni losowym przeszukiwaniu przestrzeni rozwiązań. Algorytm z pełną dywersyfikacją nie skupia się w ogóle na poszukiwaniu lokalnego minimum

Tabu Search w swojej podstawowej wersji posiada zbyt małą dywersyfikację. Mimo posiadania listy tabu, algorytm może utknąć w sąsiedztwie lokalnego minimum.

W programie realizowanym podczas projektu zostały zaimplementowane dwa elementy zwiększające dywersyfikację Tabu Search:

- Frequency-based memory – algorytm zapamiętuje dodatkowo karę danego ruchu. Jest ona zwiększana, gdy algorytm wykona ruch, który nie przyniósł poprawy rozwiązaniu, a następnie używana podczas wybierania najlepszego ruchu w sąsiedztwie.
- Dodatkowo zaimplementowany został mechanizm, który przy braku lokalnej poprawy losuje nowe rozwiązanie, które będzie używane podczas dalszego działania algorytmu.

4 IMPLEMENTACJA

Kod programu został podzielony na segmenty

- Algorithms – algorytmy zaimplementowane w programie
- Neighborhoods – sąsiedztwa zaimplementowane w programie
- States – stany w których aplikacja może się znajdować (różne poziomy menu)
- Structures – struktury wykorzystywane podczas działania programu

Z uwagi na rozmiar kodu aplikacji, zostaną omówione tylko te elementy związane z obecnym etapem projektu.

4.1 IMPLEMENTACJA SĄSIEDZTWA

Aby umożliwić proste tworzenie kolejnych sąsiedztw oraz ułatwić zmianę sąsiedztwa podczas działania programu, zostały one zaimplementowane przez zastosowanie polimorfizmu. Każda z klas sąsiedztw dziedziczy po klasie abstrakcyjnej ANeighborhood, oraz implementuje jej metody:

- Specyficzne dla każdego z algorytmów:
  - getType – zwrócenie typu sąsiedztwa, aby umożliwić jego łatwą identyfikację
  - getLoos – zwrócenie wartości ważonej sumy opóźnień dla obecnie przetwarzanego sąsiedztwa
- Specyficzne dla algorytmu SA:
  - draw – losowanie sąsiada dla podanej kolejności zadań
  - applyChanges – zmiana przekazanej kolejności zadań na sąsiada przechowywanego obecnie w pamięci
  - assignComplexity – obliczenie wartości złożoności sąsiedztwa na podstawie przekazanego rozmiaru problemu
- Specyficzne dla algorytmu SA:
  - init – inicjalizacja sąsiedztwa (przykładowo ustawienie parametrów wskazujących na początek sąsiedztwa)
  - getNext – zmiana sąsiada na kolejnego w przestrzeni $N(x_0)$. Jeżeli zostali przejrzani już wszyscy sąsiedzi, zostanie zwrócona wartość false
  - saveCurrent – zapisanie w pamięci obecnie przetwarzanego sąsiada (przykładowo jeżeli jego wynik poprawia wynik z punktu widzenia algorytmu)
  - refreaschTabu – zmniejszenie kadencji wszystkich elementów przechowywanych w liście tabu
  - increaseTabu – ustawienie kadencji dla obecnie przechowywanego w pamięci sąsiada
  - increasePunishment – zwiększenie kary dla sąsiada (dywersyfikacja)
  - getLoosWithPunish – zwrócenie wartości ważonej sumy opóźnień z uwzględnieniem kary za pogorszenia wyniku (dywersyfikacja)
  - applyBest – zmiana przekazanej kolejności zadań na najlepszego sąsiada przechowywanego obecnie w pamięci
  - notInTabu – test, czy obecnie przetwarzany sąsiad znajduje się na liście tabu
  - punischClear – czyszczenie lisy kar za pogorszenia (dywersyfikacja)

4.2 Implementacja Symulowania Wyżarzania

Symulowanie wyżarzania zostało zaimplementowane w klasie SimulatedAnnealing. Dla algorytmu został wybrany geometryczny schemat chłodzenia. Najważniejsze elementy znajdujące się w klasie to:

- setFirstTemperature – ustawienie temperatury początkowej
- setNextTemperature – ustawienie kolejnej temperatury na podstawie tej obecnie panującej
- shouldAssignCurrent – zwrócenie informacji o tym, czy obecnie przetwarzany sąsiad ma być zaakceptowany jako kolejność zadań przetwarzana w kolejnej iteracji algorytmu
- run – uruchomienie pętli głównej algorytmu

Algorytm zostanie zatrzymany po spełnieniu się jednego z dwóch warunków:

- Osiągnięcie temperatury minimalnej ustawionej w statycznym atrybucie temperatureMinimum.
- Osiągnięcie czasu działania algorytmu powyżej tej ustawionej podczas ustalania parametrów algorytmu.

Temperatura końcowa została w programie ustawiona na wartość równą 0.01, dla której prawdopodobieństwo zaakceptowanie wartości nawet o 1 pogarszającej dotychczasowy wynik jest równe $P = 3.72008 \times 10^{-44}$. Podstawa a schematu geometrycznego została ustawiona na wartość równą 0.99. Temperatura początkowa była zmieniana w trakcie prowadzenia pomiarów, z powodów opisanych w punkcie 5.1.

4.3 Implementacja Tabu Search

Przeszukiwanie z zakazami zostało zaimplementowane w klasie TabuSearch. Najważniejszymi metodami klasy tej są:

- run – uruchomienie algorytmu TS bez dywersyfikacji
- runDiversify – uruchomienie algorytmu TS z dywersyfikacją

Dodatkowo ważnymi atrybutami klasy są:

- tabu – wektor dwuwymiarowy przechowujący informację na temat kadencji zakazanych ruchów
- randAfterIters – ilość iteracji bez poprawy, dla których ma zostać wylosowane nowe rozwiązanie (dywersyfikacja)
- punishment – wektor dwuwymiarowy przechowujący informację o karach za brak poprawy rozwiązania

Algorytm zostanie zatrzymany dopiero po osiągnięciu czasu ustalonego w kryterium stopu w programie.

5 EKSPERYMENT

5.1 PLAN EKSPERYMENTU

Dla wielkości problemu równego 100 uruchomione zostały podane poniżej algorytmy:

- Symulowanie wyżarzania z użyciem sąsiedztwa typu exchange
- Symulowane wyżarzanie z użyciem sąsiedztwa typu insert
- Tabu Search bez dywersyfikacji z użyciem sąsiedztwa typu exchange
- Tabu Search bez dywersyfikacji z użyciem sąsiedztwa typu insert
- Tabu Search z dywersyfikacją z użyciem sąsiedztwa typu exchange
- Tabu Search z dywersyfikacją z użyciem sąsiedztwa typu insert

Testy zostały uruchomione dla wszystkich dostępnych na stronie http://people.brunel.ac.uk/ wygenerowanych problemów wielokości 100 pomniejszonych o te, dla których wynik sumy ważonego opóźnienia jest równa 0, dla których obliczenie błędu względnego nie miałoby sensu. Całkowita ilość danych testowanych wyniosła 107. Wartość błędu względnego została obliczona na podstawie wzoru:

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

Następnie wszystkie wyniki zostały uśrednione, aby zmniejszyć działanie sytuacji losowych, związanych ze specyfiką działania algorytmów.

Na takiej zasadzie algorytmy zostały uruchomione dla kolejnych czasów wykonywania się t równych 1s, 2s, 5s, 10s. Dodatkowo dla algorytmu SA wartość temperatury początkowej była zwiększana proporcjonalnie do ustawionego kryterium stopu w taki sposób, aby algorytm był w stanie wykonywać się zadeklarowaną ilość czasu. Powodem jest fakt, że gdyby ustawić temperaturę na odpowiednio wysoką dla wszystkich czasów wykonywania się, w przypadku przerwania algorytmu po 1s byłby on w stanie dużej dywersyfikacji, co nie pozwoliłoby mu na odnalezienie lokalnego minimum. Z uwagi na to, że dla algorytmu TS podczas każdej iteracji wymagany jest pełen przegląd sąsiedztwa, dla wszystkich jego czterech wariantów został uruchomiony dodatkowy test trwający 20 sekund dla każdej instancji.

5.2 SPOSÓB POMIARU CZASU

Pomiar czasu odbywa się z użyciem std::chrono::high_resolution_clock. Klasa ta pozwala na wyznaczanie bardzo małych odstępów czasu. Przykład wyznaczania odstępu czasu:

```
std::chrono::high_resolution_clock::time_point start = std::chrono::high_resolution_clock::now();
//obliczenia, których czas chcemy zmierzyć
std::chrono::high_resolution_clock::time_point end = std::chrono::high_resolution_clock::now();
long long time = std::chrono::duration_cast<std::chrono::microseconds>(end - this->start).count();
```

Wynikiem jest czas podany w mikrosekundach.

5.3 WYNIKI EKSPERYMENTU

5.3.1 Symulowanie wyżarzania

5.3.1.1 Sąsiedztwo przez zamianę

Średni czas wykonywania	**Średnia wartość błędu względnego**
[ms]	[%]
988.322	0.0062
1998.578	0.0073
4960.290	0.0013
9426.335	0.0011

Tabela 1 Symulowanie wyżarzania - sąsiedztwo przez zamianę

5.3.1.2 Sąsiedztwo przez wstawianie

Średni czas wykonywania	**Średnia wartość błędu względnego**
[ms]	[%]
1002.117	0.0233
2002.669	0.0279
5002.617	0.0079
10002.375	0.0103

Tabela 2 Symulowanie wyżarzania - sąsiedztwo przez wstawianie

5.3.1.3 Zmiana początkowej temperatury względem ustawionego kryterium stopu

Kryterium stopu [s]	**Początkowa temperatura**
1	0.12
2	2
5	10 000
10	$1 * 10^9$

Tabela 3 Zmiana początkowej temperatury względem ustawionego kryterium stopu

5.3.2 Tabu Search

5.3.2.1 Dywersyfikacja, sąsiedztwo przez zamianę

Średni czas wykonywania	Średnia wartość błędu względnego
[ms]	[%]
1000.598	38.9552
2000.697	24.5077
5000.673	1.2990
10000.703	0.0311
20000.698	0.0255

Tabela 4 Tabu Search - dywersyfikacja, sąsiedztwo przez zamianę

5.3.2.2 Brak dywersyfikacji, sąsiedztwo przez zamianę

Średni czas wykonywania	Średnia wartość błędu względnego
[ms]	[%]
1000.724	36.6471
2000.743	26.7255
5000.645	0.2759
10000.684	0.0897
20000.672	0.1363

Tabela 5 Tabu Search - brak dywersyfikacji, sąsiedztwo przez zamianę

5.3.2.3 Dywersyfikacja, sąsiedztwo przez wstawianie

Średni czas wykonywania	Średnia wartość błędu względnego
[ms]	[%]
1001.624	81.4347
2001.581	55.3770
5001.376	20.0105
10001.483	0.7245
20001.515	0.0815

Tabela 6 Tabu Search - dywersyfikacja, sąsiedztwo przez wstawianie

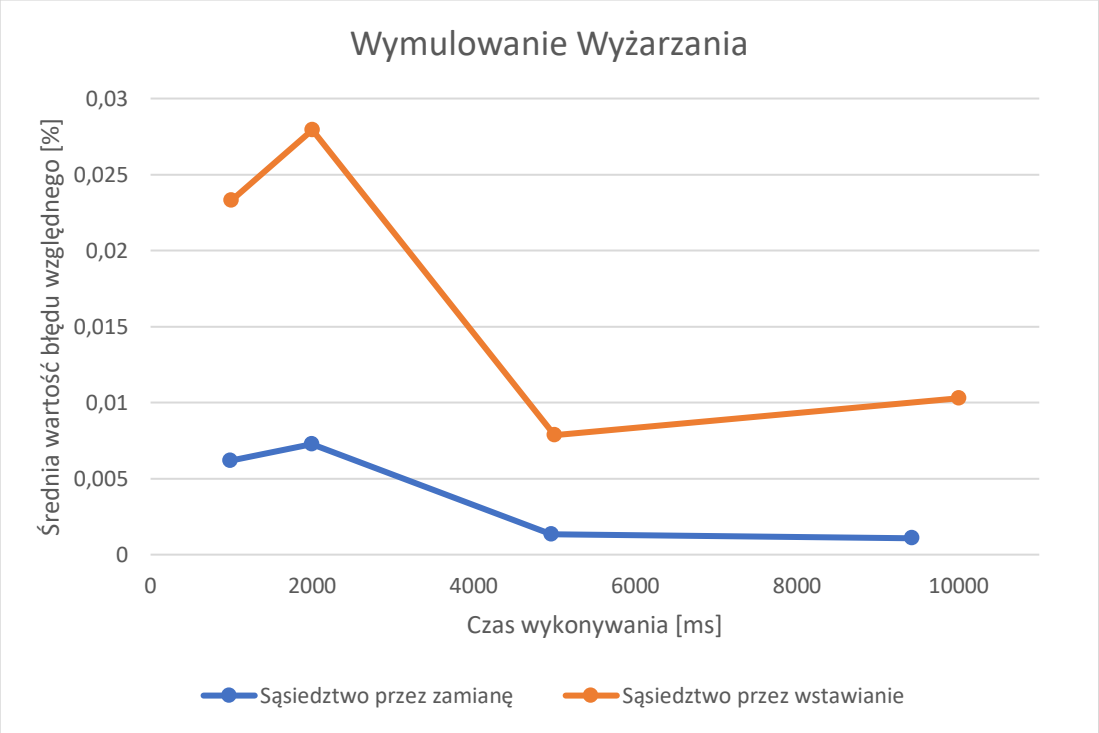
5.3.2.4 Brak dywersyfikacji, sąsiedztwo przez wstawianie

Średni czas wykonywania	Średnia wartość błędu względnego
[ms]	[%]
1001.417	82.9170
2001.636	84.1956
5001.455	24.5465
10001.408	0.7728
20001.522	0.1122

Tabela 7 Tabu Search - brak dywersyfikacji, sąsiedztwo przez wstawianie

5.4 ANALIZA WYNIKÓW

5.4.1 Symulowanie wyżarzania



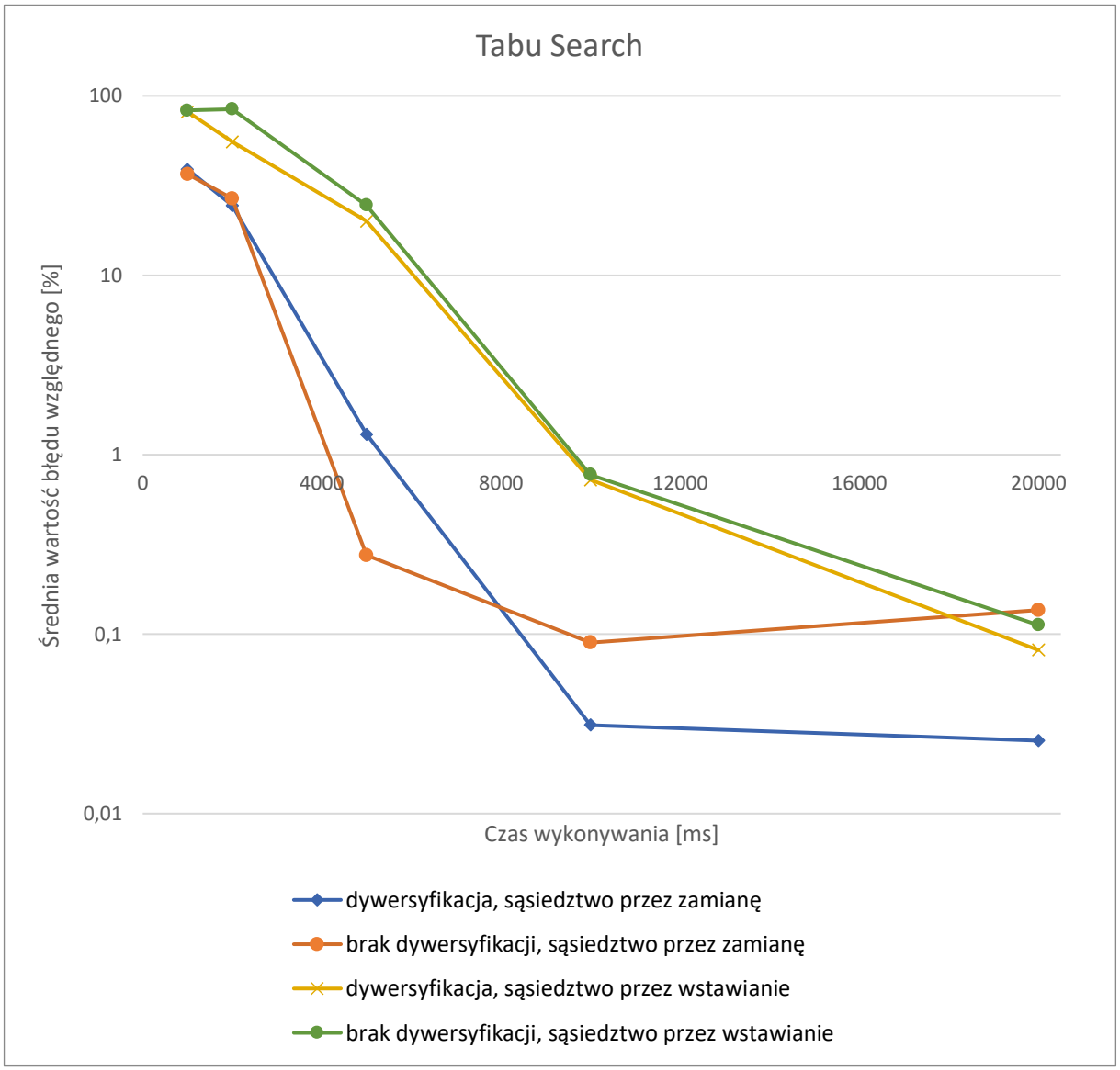
Wykres 1 Symulowanie Wyżarzania

Na powyższym wykresie można zauważyć, że dla Symulowanego Wyżarzania lepsze wyniki osiągnięte zostały przez sąsiedztwo typu przez zamianę. Powodem może być większy czas wykonywania się operacji przeniesienia indeksu elementu (algorytm był przerywany, jeżeli wykonywał się zbyt długą ilość czasu, różnica wykonywania się algorytmu między sąsiedztwami wynosiła dla 10 s około 0.5s). Innym powodem może być podobny sposób generowania wszystkich struktur. Spodziewane czasy zakończenia zadań, ich wagi oraz same czasy wykonywania były generowane z użyciem rozkładu jednostajnego ciągłego. Zmiana sposobu generowania problemów mogłaby spowodować mniejszą ilość minimów lokalnych w sąsiedztwie przez wstawianie, co zwiększyłoby jego jakość.

Z powodu sposobu wykonywania pomiarów (wraz ze zmianą kryterium stopu zmieniana była temperatura początkowa, co można zauważyć w tabeli nr 3) można dojść do kolejnych wniosków. Wraz ze zwiększaniem się temperatury początkowej pozwalamy na początkowe częstsze uwzględnianie wyników pogarszających, co pozwala na początkowe zwiększenie dywersyfikacji algorytmu. Jak widać na wykresie 1, pozwoliło to na nieznaczną poprawę generowanych wyników algorytmu. Jednak nawet dla małej temperatury początkowej wyniki te były bliskie najlepszym znanym wynikom. Może to sugerować małą ilość minimów lokalnych wygenerowanych problemów, lub ich bardzo zbliżone wartości wyników. Warto zauważyć, że dla małej temperatury algorytm Symulowanego Wyżarzania bardzo przypomina wersję greedy algorytmu przeszukiwania lokalnego, co oznacza, że algorytm ten mógłby być całkiem wydajny dla problemów generowanych w podobny sposób.

5.4.2 Tabu Search

Z uwagi na dużą różnicę błędu względnego między mniejszymi i większymi kryteriami stopu, wyniki na wykresie zostały przedstawione w skali logarytmicznej.



Wykres 2 Tabu Search

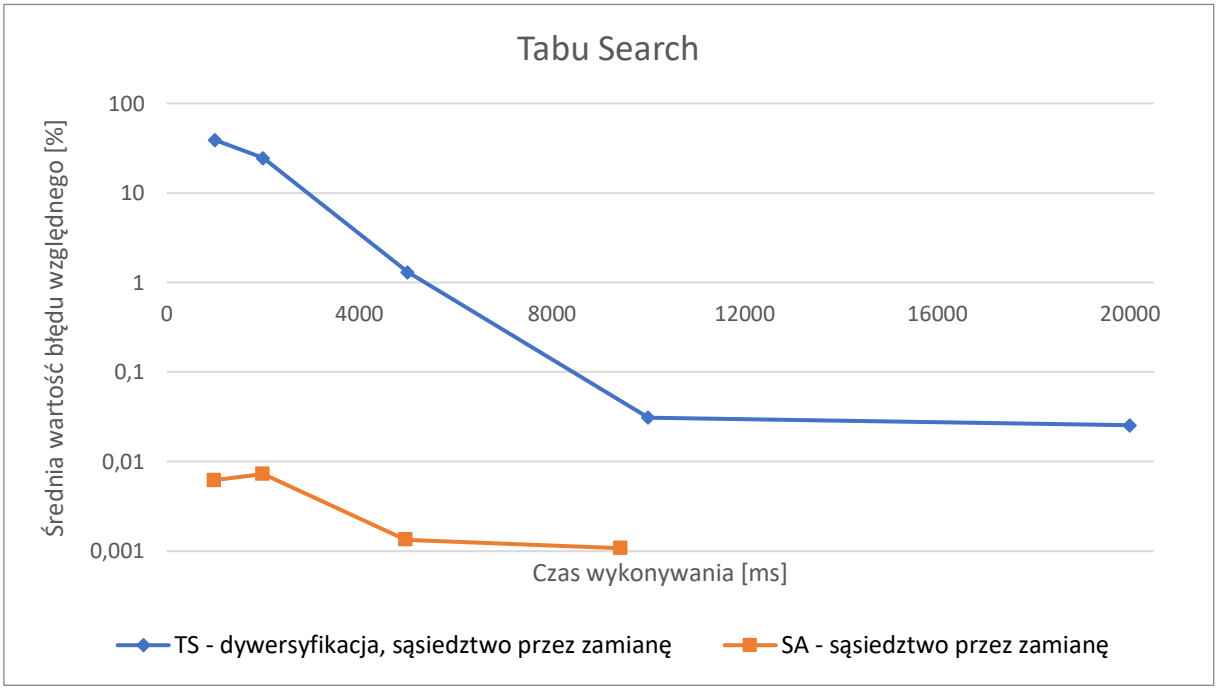
Na powyższym wykresie można zauważyć jak ogromną początkową różnicę daje zwiększanie kryterium stopu (czasu wykonywania się algorytmu). Powodem może być ilość elementów, które należy przejrzeć dla danego sąsiedztwa (dla badanej wielkości problemu wynosi ono odpowiednio około 4 850 iteracji dla sąsiedztwa przez zamianę oraz 9 900 dla sąsiedztwa przez wstawianie). Dla małej wartości kryterium stopu algorytm często nie jest wstanie znaleźć nawet pierwszego minimum lokalnego.

Różnice między wersją algorytmu z dywersyfikacją oraz bez niej są zauważalne dopiero przy większych wartościach kryterium stopu. Szczególnie różnicę tą widać w sąsiedztwie przez zamianę. Większa dywersyfikacja algorytmu pozwala na większą szansę odwiedzenia dużej ilości minimów lokalnych.

Zauważalna różnica jest również między typami sąsiedztw. Oprócz przypuszczalnych powodów przedstawionych już podczas opisu Symulowania Wyżarzania, dodatkowo elementem decydującym mogła być różnica w rozmiarze sąsiedztw, który dla sąsiedztwa przez zamianę była mniejsza. Sprawiło to, że algorytm TS dla tego sąsiedztwa szybciej był w stanie osiągać minima lokalne. Jednak różnica ta mogłaby zostać zmniejszona przy dłuższym czasie wykonywania się algorytmów. Niestety pomiary dla 20 sekund wszystkich problemów wielkości 100 zadań zajmowała już około 2 godzin, a testy wykonane na jednym problemie mogłyby zwiększyć czynnik losowy wyników.

6 WNIOSKI

Podczas wykonywania projektu zostały zaimplementowane dwa algorytmy: Symulowane Wyżarzanie oraz Tabu Search. Na poniższym wykresie zostało przedstawione zestawienie wyników dla wersji dających najlepsze rozwiązanie dla największego ustawionego kryterium stopu:



Wyniki pokazują przewagę algorytmu SA nad algorytmem TS. Jednak należy zauważyć, że dalsze zmiany wartości temperatury początkowej w algorytmie SA mogą nie przynosić dużych korzyści, jednak algorytm TS pozostawiony na dłuższy czas działania może przynosić bardziej korzystne wyniki.

Algorytm SA można zmodyfikować w taki sposób, aby mógł wykonywać się dłuższą ilość czasu, przykładowo przez dodanie elementu zwiększającego temperaturę, jeżeli wariancja kolejnych wyników jest odpowiednio niska.

Wadą zaimplementowanego algorytmu TS jest przeglądanie pełnego sąsiedztwa, co generuje długi czas poszukiwania minimum lokalnego. Można temu przeciwdziałać poprzez zastosowanie dodatkowego kryterium aspiracji o nazwie „Aspiracja plus”. Dodaje ono możliwość przeszukiwania danego sąsiedztwa aż do osiągnięcia pewnej wartości progowej, co pozwoli na szybsze osiąganie przez algorytm minimów lokalnych.