Optymalizacja kombinatoryczna
Algorytm aproksymacyjny

Bartosz Kukawka (75911)
Marcin Mikołajczak (75922)
Grupa I3A

29 listopada 2006

Spis treści

1 Opis problemu

1.1 Notacja trójpolowa

Celem projektu było opracowanie algorytmu znajdującego rozwiązania możliwie bliskie optymalnym dla optymalizacyjnego problemu szeregowania zadań określonego w notacji trójpolowej następująco:

$$F2, h_{1k} \mid SRS\ , r_i, no-wait \mid C_{max}$$

Poszczególne symbole opisu precyzują problem w następujący sposób:

$F2$ – szeregowanie zadań odbywa się w systemie przepływowym na 2 maszynach (każde zadanie przetwarzane jest dwufazowo – najpierw na I maszynie, potem na II maszynie),

h_{1k} – na I maszynie występują okresy niedostępności,

SRS – istnieje możliwość wznawiania zadań (z zastrzeżeniem, że zatrzymanie zadania może nastąpić jedynie w przypadku wystąpienia okresu niedostępności, ponadto po wznowieniu konieczne jest ponowne wykonanie prac przygotowawczych),

r_i – dane są momenty przybycia zadań,

$no-wait$ – przetwarzanie na maszynie II musi rozpocząć się natychmiast po zakończeniu przetwarzania na maszynie I,

C_{max} – kryterium optymalizacyjne stanowi moment zakończenia przetwarzania ostatniego zadania (możliwie jak najwcześniej).

1.2 Opis wejścia

Opis instancji problemu sprowadza się do podania następujących danych:

n – liczba szeregowanych zadań,

- r'_i – czas przybycia zadania,
- s'_i – czas przygotowania na I maszynie,
- p'_i – czas przetwarzania na I maszynie,
- s''_i – czas przygotowania II maszynie,
- p''_i – czas przetwarzania na II maszynie (dla $i = 1, 2, \ldots, n$);

m – liczba okresów niedostępności,

- r_j – czas rozpoczęcia okresu niedostępności,
- p_j – długość okresu niedostępności dla ($j = 1, 2, \ldots, m$).

Dziedzinę problemu stanowią liczby całkowite nieujemne, tj. $n, m \in \mathbb{N}$ oraz r'_i, s'_i, p'_i, s''_i, p''_i, r_j, $p_j \in \mathbb{N}$ dla wszystkich dopuszczalnych wartości i oraz j.

Dodatkowo problem został ograniczony ze względu na wartości występujące w instancji, które muszą spełniać następujące warunki:

- dla każdego zadania czasy przetwarzania na obu maszynach zawierają się w przedziale 1-100 jednostek, czyli:

$$\forall i = 1, 2, \ldots, n: \ 1 \leq p'_i \leq 100 \ \wedge \ 1 \leq p''_i \leq 100$$

- dla każdego zadania czas przygotowania na danej maszynie nie przekracza 50% czasu przetwarzania na tej maszynie, czyli:

$$\forall i = 1, 2, \ldots, n: \ 1 \leq s'_i \leq 50\,\%\, p'_i \ \wedge \ 1 \leq s''_i \leq 50\,\%\, p''_i$$

- czasy trwania wszystkich okresów niedostępności zawierają się w przedziale 1-100 jednostek, czyli:

$$\forall j = 1, 2, \ldots, m: \ 1 \leq p_j \leq 100$$

oraz ze względu na sumaryczne czasy przestojów i przetwarzania:

$$\sum_{j=1}^{m} p_j \leq 20\,\% \sum_{i=1}^{n} (s'_i + p'_i)$$

W dalszych rozważaniach wygodniejsze będzie posługiwanie się jedynie wartościami n i m, dlatego powyższa zależność zostanie odpowiednio przekształcona z wykorzystaniem teorii prawdopodobieństwa tak, aby uwzględniała losową naturę generowanych testów.

Niech zmienna losowa X odpowiada czasowi przetwarzania na I maszynie, zmienna losowa Y czasowi przygotowania na II maszynie, a zmienna Z długości okresu niedostępności.

Zmienna X przyjmuje wartości x_i dla $i = 1, 2, \ldots, 99$, gdzie x_i odpowiada czasowi przetwarzania równemu $i + 1$ jednostek czasu. Rozkład zmiennej X jest równomierny, zatem:

$$\forall i = 1, 2, \ldots, 99: \ \mathrm{P}(X = x_i) = \frac{1}{99}$$

Wartość średnią zmiennej X można obliczyć wg wzoru:

$$\mathrm{E}(X) = \sum_{i=1}^{99} x_i \mathrm{P}(X = x_i) = 51$$

Zmienna Y przyjmuje wartości y_k dla $k = 1, 2, \ldots, 50$, gdzie y_k odpowiada czasowi przetwarzania równemu k jednostek czasu. Rozkład zmiennej Y jest równomierny, zatem:

$$\mathrm{P}(Y = y_k | X = x_i) = \begin{cases} \frac{1}{x_i/2} & k \leq x_i/2 \\ 0 & k > x_i/2 \end{cases}$$

Wartość średnią zmiennej Y można obliczyć korzystając z następujących zależności:

$$\mathrm{E}(Y|X) = \int_{-\infty}^{\infty} y \ \mathrm{dF}(y|x) = \sum_{k=1}^{50} \mathrm{P}(Y = y_k | X = x_i)$$

$$\mathrm{E}(Y) = \int_{-\infty}^{\infty} \mathrm{E}(Y|X)\ \mathrm{dF}_1(x) = \sum_{i=1}^{99} \mathrm{E}(Y|X = x_i)\mathrm{P}(X = x_i)$$

Końcowy wzór ma zatem postać:

$$\mathrm{E}(Y) = \sum_{i=1}^{99}\sum_{k=1}^{50} \mathrm{P}(Y = y_k|X = x_i)\mathrm{P}(X = x_i) = 13{,}13$$

Zmienna Z przyjmuje wartości Z_i dla $i = 1, 2, 3, \ldots, 100$, gdzie z_i odpowiada czasowi niedostępności równemu i jednostek czasu. Rozkład zmiennej Z jest równomierny, zatem:

$$\mathrm{P}(Z = z_i) = \frac{1}{100}$$

Wartość średnią zmiennej Z można obliczyć wg wzoru:

$$\mathrm{E}(Z) = \sum_{i=1}^{100} z_i \mathrm{P}(Z = z_i) = 50{,}5$$

Z wartości średnich zmiennych losowych X, Y, Z można wyznaczyć oczekiwany stosunek liczby zadań do liczby okresów niedostępności tak, aby spełnione były warunki zadania:

$$m\,\mathrm{E}(Z) \leq 20\,\%\,n\,(\mathrm{E}(X) + \mathrm{E}(Y))$$

Po przekształceniu i podstawieniu wartości otrzymujemy:

$$m \leq 25{,}4\,\%\,n$$

1.3 Opis wyjścia

Dla danej instancji problemu rozwiązanie stanowi n opisów uszeregowań zadań, czyli zestaw następujących liczb:

l'_i – ilość okresów przetwarzania i-tego zadania na I maszynie,

r'_{ik} – czas rozpoczęcia j-tego okresu przetwarzania i-tego zadania,

p'_{ik} – długość j-tego okresu przetwarzania i-tego zadania (dla $k = 1, 2, \ldots, l'_i$);

r''_i – czas rozpoczęcia przetwarzania i-tego zadania na II maszynie (dla $i = 1, 2, \ldots, n$).

W dalszych rozważaniach przyjęto, że okresy przetwarzania na I maszynie są w obrębie danego zadania posortowane chronologicznie, tj. zgodnie z rosnącą wartością r'_{ik}:

$$\forall i = 1, 2, \ldots, n\, \forall j = 1, 2, \ldots, l'_i - 1 :\ r'_{ij} < r'_{i(j+1)}$$

Przy tak przyjętym opisie rozwiązania zachodzi równość:

$$C_{max} = \max_{i=1}^{n} r''_i + p''_i$$

Zgodnie z definicją postawionego problemu liczby wymienione powyżej muszą spełniać następujące warunki:

- przetwarzanie każdego z zadań rozpoczyna się najwcześniej w momencie jego przybycia (rozważamy pierwszy okres przetwarzania tego zadania na maszynie I), czyli:

$$\forall i = 1, 2, \ldots, n : p'_{i1} \geq r'_i$$

- dla każdego zadania sumaryczna długość okresów pracy zaplanowanych na I maszynie musi być równa czasowi zadanemu w instancji, czyli:

$$\forall i = 1, 2, \ldots, n : \sum_{k=1}^{l'_i} p'_{ik} = p'_i$$

- dla każdego zadania przetwarzanie na maszynie II musi rozpocząć się natychmiast po zakończeniu przetwarzania na maszynie I (rozważamy ostatni okres przetwarzania tego zadania na maszynie I), czyli:

$$\forall i = 1, 2, \ldots, n : r'_{il'_i} + p'_{il'_i} = r''_i$$

- niemożliwe jest przetwarzanie zadania podczas okresu niedostępności oraz jednoczesne przetwarzanie więcej niż jednej części tego samego zadania lub różnych zadań, innymi słowy dla zbioru okresów przetwarzania:

$$A = \{(i, j) : \ i = 0, 1, \ldots, n \ \wedge \ j = 1, 2, \ldots, l'_i\}$$

  gdzie jako zadanie o numerze $i = 0$ przyjmujemy dodatkowe zadanie o okresach przetwarzania pokrywających się z okresami niedostępności, czyli:

$$s'_0 = 0$$
$$l'_0 = m$$
$$\forall j = 1, 2, \ldots, m : r'_{0j} = r_j \wedge p'_{0j} = p_j$$

  okresy zawarte w zbiorze A odpowiadają parami rozłącznym przedziałom czasu, czyli:

$$\forall (i_1, j_1), (i_2, j_2) \in A, (i_1, j_1) \neq (i_2, j_2) :$$
$$[r'_{i_1 j_1} - s'_{i_1 j_1}, r'_{i_1 j_1} + p'_{i_1 j_1}) \cap [r'_{i_2 j_2} - s'_{i_2 j_2}, r'_{i_2 j_2} + p'_{i_2 j_2}) = \emptyset$$

- dla określonej kolejności zadań przetwarzanie każdego z nich zaczyna się w pierwszej możliwej chwili czasu, tj. nie występują opóźnienia inne, niż wynikające z opisanych wyżej warunków.

Wszystkie z powyższych warunków były automatycznie sprawdzane dla każdego rozwiązania wygenerowanego w fazie testów poprawnościowych projektu.

2 Opis algorytmu

2.1 Algorytm siłowy

Siłowe rozwiązanie postawionego problemu wymaga sprawdzenia wartości kryterium C_{max} dla wszystkich możliwych kolejności wykonania zadań, czyli wszystkich permutacji n - elementowego ciągu, co daje algorytm o złożoności rzędu $O(n!)$. W obrębie danej kolejności zadań rozpatrywane jest jedynie uszeregowanie o najmniejszej wartości C_{max}. Algorytm typu *brute force* został zaimplementowany jako pierwszy, jako narzędzie umożliwiające sprawdzenie poprawności algorytmu obliczającego wartość C_{max}. W praktyce algorytm taki ma rozsądny czas działania dla instancji o liczbie zadań $n \leq 10$, dlatego w dalszych rozważaniach zostanie on pominięty.

W implementacji algorytmu siłowego wydzielona została część odpowiedzialna za obliczanie wartości C_{max}, w celu jej późniejszego wykorzystania w algorytmie aproksymacyjnym. Obliczenie wspomnianego parametru wymaga wyznaczenia momentu zakończenia każdego z n zadań, co z kolei wymaga wyszukania wśród m okresów niedostępności tych, które przerywają wykonywanie zadania. Ponieważ implementacja algorytmu wykorzystuje liniowo uporządkowane struktury tablicowe do przechowywania opisów zarówno zadań jak i okresów niedostępności, jej złożoność obliczeniowa jest rzędu $O(n + m)$. Przy założeniu, że $m \leq 25{,}4\,\% n$, złożoność wynosi $O(n)$.

2.2 Algorytm aproksymacyjny

2.2.1 Idea algorytmu

Do rozwiązania postawionego problemu wykorzystany został schemat algorytmu typu *tabu search*. Ogólna idea tego algorytmu polega na iteracyjnym poprawianiu pewnego rozwiązania, ustalonego na początku działania algorytmu. Rozwiązanie to może być wylosowane, oszacowane przy pomocy algorytmu zachłannego lub wyliczone jako pierwsze (np. w sensie leksykograficznym) dozwolone w warunkach zadania, przy czym, im bliższe jest ono rozwiązaniu optymalnemu, tym szybciej i z lepszymi efektem zakończy się działanie algorytmu. Proces poprawiania rozwiązania polega na sekwencyjnym wykonywaniu na nim pewnej operacji, zwanej *krokiem elementarnym*, która prowadzi do innego rozwiązania. Zbiór wszystkich rozwiązań, które można otrzymać przez wykonanie dowolnego z możliwych kroków elementarnym nazywane jest sąsiedztwem rozwiązania. Krokiem wykonywanym w danej iteracji jest ten, spośród możliwych, który prowadzi do rozwiązania najlepszego w aktualnym sąsiedztwie. Po wykonaniu operacji jest ona umieszczana na liście kroków zabronionych, zwanej *listą tabu*, co ma na celu wykluczenie sytuacji, w której algorytm zapętli się, wykonując naprzemiennie te same kroki. Krok znajdujący się na liście tabu może zostać wykonany jedynie wtedy, kiedy spełnia pewien dodatkowy warunek, zwany *kryterium aspiracji*. Warunek stopu określa się w implementacji algorytmu.

2.2.2 Implementacja algorytmu

Program rozwiązujący postawiony problem został podzielony na dwie części: procedury obliczające optymalny sposób przetwarzania zadań dla pewnej ustalonej ich kolejności (permutacji) oraz część związaną z poszukiwaniem tej kolejności, dla której wartość C_{max} będzie

jak najmniejsza. W ramach pierwszej części wykorzystany został algorytm opracowany na potrzeby rozwiązania siłowego, druga została zaimplementowana z wykorzystaniem idei algorytmu *tabu search*, którego elementy sprecyzowano w opisany niżej sposób.

Krok elementarny W fazie doświadczalnej projektu testowane były algorytmy korzystające z trzech różnie zdefiniowanych kroków elementarnych. Wersje te, zwane dalej *MoveIndex*, *SwapIndex* i *SwapTask*, operują na permutacji wyznaczającej kolejność przetwarzania zadań i realizują krok elementarny opisany parą liczb (a, b) w następujący sposób:

MoveIndex – przemieszczenie zadania z pozycji a na pozycję b,

SwapIndex – zamiana miejscami zadań na pozycjach a i b,

SwapTask – zamiana miejscami zadań o numerach a i b.

Sprawdzenie każdego z kroków wymaga obliczenia wartości C_{max} dla powstałej permutacji zadań, czyli wykonania – zgodnie z wcześniejszymi ustaleniami – $O(n)$ operacji. Ilości możliwych kroków oraz wynikającą z nich złożoność obliczeniową algorytmu dla poszczególnych wersji przedstawia tabela 1.

Metoda	Ilość kroków	Złożoność
MoveIndex	$(n-1)^2$	$O(n^3)$
SwapIndex	$\frac{n(n-1)}{2}$	$O(n^3)$
SwapTask	$\frac{n(n-1)}{2}$	$O(n^3)$

Tablica 1: Ilość kroków i złożoność dla poszczególnych wersji algorytmu

Lista tabu Wykonanie kroku elementarnego wiąże się z jego umieszczeniem na liście tabu. Początkowo listę tabu zaimplementowano jako drzewo binarne. Ostatecznie, w celu poprawienia wydajności, opisy kroków umieszczono w strukturze danych umożliwiającej dodawanie, usuwanie oraz sprawdzanie zawierania elementów w czasie stałym. W fazie doświadczalnej projektu testowane były dwie koncepcje listy tabu, zwane dalej *DoubleBlock* i *MultiBlock*, które dla kroku opisanego przez parę liczb (a, b) powodują zablokowanie:

DoubleBlock – danego kroku i kroku odwrotnego, tj. kroków (a, b) i (b, a),

MultiBlock – wszystkich kroków zawierających w opisie liczby a i b, czyli takich kroków (x, y) i (y, x), że $x \neq y$ i $x \in \{a, b\}$, a y jest dowolną liczbą tworzącą razem z x prawidłowy krok.

Długość listy tabu jest parametrem algorytmu (zwanym dalej *tabus*). W szczególności, dla długości równej 0, implementacja staje się realizacją algorytmu *local search*.

Warunek stopu Iteracje algorytmu wykonywane są bezwarunkowo dopóty, dopóki poprawia się wartość C_{max}. Następnie wykonywana jest pewna ilość dodatkowych iteracji, w trakcie których możliwa jest dalsza zmiana sąsiedztwa rozwiązania, aż do ewentualnego znalezienia obszaru o lepszej wartości C_{max}. W przypadku nie znalezienia takiego obszaru algorytm kończy się. Sumaryczna ilość dodatkowych iteracji jest parametrem algorytmu (zwanym dalej *chances*).

Kryterium aspiracji Aby zwiększyć skuteczność algorytmu wprowadzono dodatkowe kryterium umożliwiające wykonanie ruchu w sytuacji, gdy znajduje się on na liście tabu. Jest to możliwe w przypadku, kiedy wykonanie zabronionego ruchu prowadzi do uzyskania rozwiązania lepszego niż najlepsze dotychczas znalezione.

2.2.3 Usprawnienia algorytmu

Ograniczenie liczby ruchów W celu ograniczenia złożoności algorytmu wprowadzony został dodatkowy parametr, zwany dalej *distance*. Mówi on, jaka może być maksymalna odległość w permutacji pomiędzy zadaniem, a miejscem w które zostanie przesunięte w kroku elementarnym. W szczególności, dla odległości równej 1, rozważane są jedynie sąsiednie zadania, co daje złożoność algorytmu rzędu $O(n^2)$.

Dokładne wzory na ilość możliwych ruchów w zależności od liczby zadań i parametru *distance* są następujące:

- dla implementacji *SwapIndex* i *SwapTask*:

$$\sum_{i=0}^{n-1} \min\{\textit{distance}, n-i-1\}$$

- dla implementacji *MoveIndex*:

$$\sum_{i=0}^{n-1} (\min\{\textit{distance}, i\} + \min\{\textit{distance}, n-i-1\}) + 1$$

Zachłanna inicjalizacja W pierwszej implementacji początkowa permutacja była losowana. Metoda ta nie sprawdza się jednak przy niezerowych czasach przybycia, gdyż prowadzi do rozwiązań dalekich od optymalnych. Znacznie lepsze efekty przynosi wstępne posortowanie zadań rosnąco według ich momentów przybycia, czyli rozważanie pewnego „naturalnego", chronologicznego porządku zadań.

Dalsze usprawnienie algorytmu możliwe jest dzięki wstępnemu ustawieniu zadań w sposób zachłanny, tj. kolejno wybierając zadania, które w danym momencie prowadzą do najkrótszego uszeregowania.

3 Wyniki eksperymentu

Faza doświadczalna opierała się na serii zautomatyzowanych testów dokonywanych na losowo generowanych instancjach. Pomiary wykonywano w środowisku GNU/Linux przy pomocy kompilatora GCC w wersji 4.1.2 na komputerach z procesorami AMD Athlon XP $1{,}8\,\mathrm{GHz}$ i Intel Celeron M $1{,}4\,\mathrm{GHz}$. Podczas analizy pomiary wykonane na różnych komputerach nie były wzajemnie porównywane.

Pomiaru czasu dokonywano systemowym poleceniem *time*, odnotowując jedynie czas spędzony przez proces w trybie użytkownika, zmniejszając w ten sposób wpływ operacji wejścia/wyjścia na wyniki pomiaru, który dla szybko wykonujących się testów mógł być znaczący. Czas mierzono w sekundach, z dokładnością do drugiego miejsca po przecinku. Większa dokładność na typowym systemie klasy PC jest w praktyce nieosiągalna.

Jakość rozwiązania mierzono porównując błędy względne pomiarów, oznaczane dalej jako *cmax_error*, przy czym jako wartość dokładną przyjmowano najmniejszą wartość C_{max} osiągniętą w trakcie wykonywania pomiarów dla danej instancji:

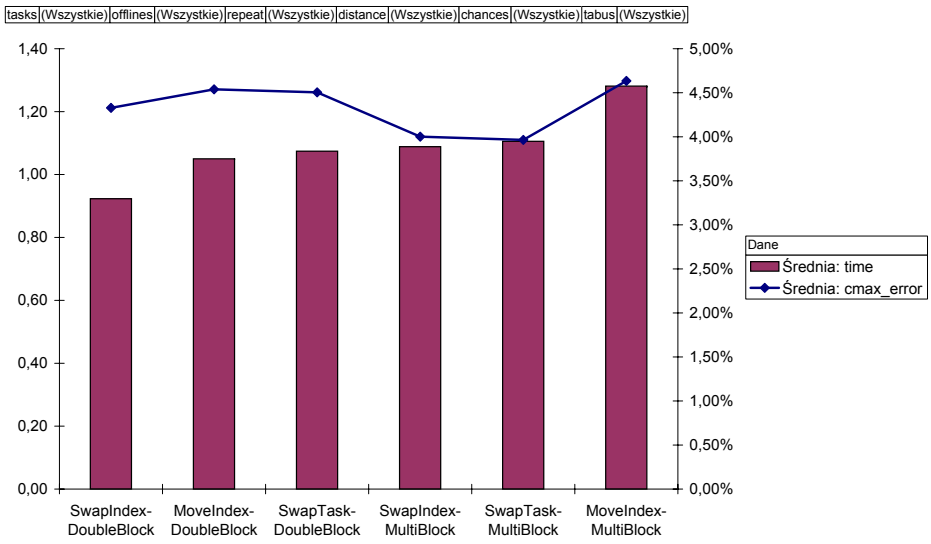
$$\forall p \in \mathrm{P}(\mathbb{I}) : \mathit{cmax\_error}(p) = \frac{\mathit{cmax}(p)}{\min_{q \in \mathrm{P}(\mathbb{I})} \mathit{cmax}(q)} - 1$$

gdzie $\mathrm{P}(\mathbb{I})$ oznacza zbiór pomiarów wykonanych dla pewnej instacji $\mathbb{I}$.

W celu zwiększenia pewności pomiarów, instancje o zadanych parametrach n i m były generowane w 20 egzemplarzach, a pomiary przeprowadzone na nich – odpowiednio uśredniane. Tam, gdzie nie jest to wyszczególnione, ilość okresów niedostępności wynosiła $m = 20\,\% \, n$.

3.1 Wybór implementacji

Pierwszym etapem eksperymentu był wybór najlepszej z opracowanych wersji algorytmu. W tym celu przeprowadzono 3 serie pomiarów, odpowiednio dla $n = 25$, $n = 50$ i $n = 75$.



Rysunek 1: Porównanie jakości i prędkości poszczególnych wersji algorytmu

Testy nie wyłoniły jednoznacznego zwycięzcy, dlatego wszystkie wyniki uśredniono do postaci przedstawionej na wykresie na rysunku 1. Wykres pokazuje, że dla przypadku średniego kryterium jakości rozwiązania również nie jest rozstrzygające. Względny błąd rozwiązań

kształtuje się w poziomie od 3,9% (wersja *SwapTask-MultiBlock*) do 4,6% (wersja *MoveIndex-MultiBlock*), zatem różnica w jakości jest minimalna. Dlatego zdecydowano się na wybór wersji najszybszej, która daje dla badanych instancji średnio ok. 30% oszczędność (czas wykonania na poziomie od 0,9 s – dla *SwapIndex-DoubleBlock* – do 1,3 s – dla *MoveIndex-MultiBlock*). Do dalszych rozważań wybrano zatem algorytm w wersji *SwapIndex-DoubleBlock*, czyli algorytm dokonujący zamian miejscami zadań o danych pozycjach, zapamiętujący na liście tabu wykonany krok i krok do niego odwrotny.

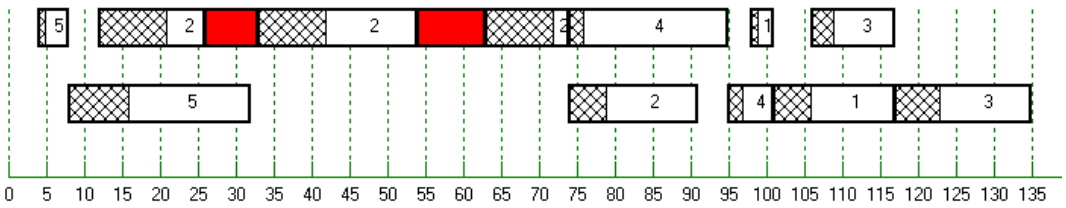
3.2 Przykład działania

Do celów demonstracyjnych wylosowano następującą instancję problemu:

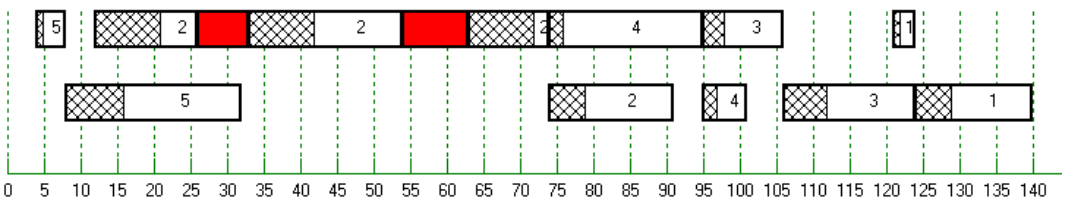
```
2
26 7
54 9
5
29 1 2 5 11
12 9 19 5 12
32 3 8 6 12
23 2 19 2 4
4 1 3 8 16
```

Algorytm wykonano z parametrami: $distance = 1$, $tabus = 3$, $chances = 5$. Przebieg działania algorytmu jest następujący:

1. Po posortowaniu według czasów przybycia powstaje wstępne uszeregowanie o funkcji celu $C_{max} = 135$:



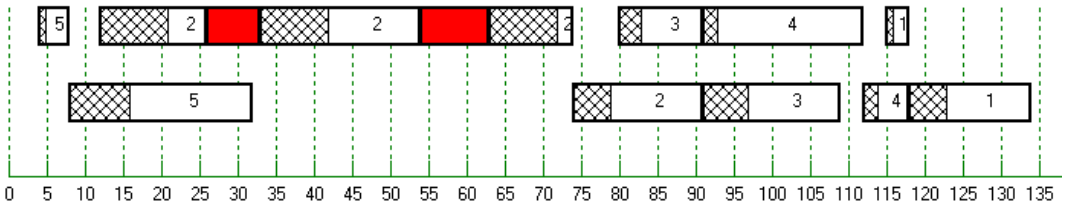
2. Lista tabu jest początkowo pusta: $\{\}$, $chances = 5$. Spośród 4 możliwych ruchów najlepszy jest ruch polegający na zamianie miejscami zadań na pozycjach 4 i 5. Ruch ten prowadzi do następującego uszeregowania:



   Do listy tabu dodany ruch $(4, 5)$ i ruch do niego odwrotny $(5, 4)$. Dla ruchu polegającego na zamianie zadań jest to dokładnie ten sam ruch, ale w ogólności nie musi tak być.

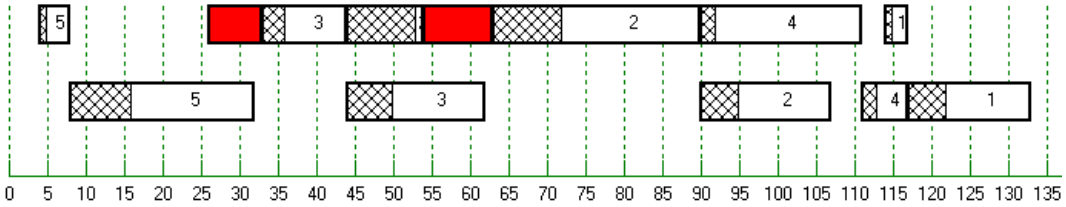
Dla implementacji *MoveIndex-DoubleBlock* i parametru *distance* > 1 ruchy te mogą być różne. W dalszej części przykładu pominięto ruchy odwrotne jako nieistotne w tym przypadku. Ruch $(4, 5)$ nie doprowadza do poprawienia wyniku zatem *chances* zostaje zmniejszone o 1 i wynosi 4.

3. Spośród 3 możliwych ruchów najlepszy jest ruch $(3, 4)$. Nie znajduje się on na liście tabu, zatem zostaje wykonany:



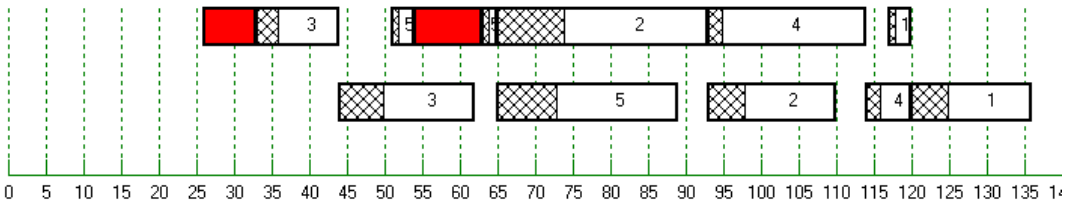
Lista tabu wygląda teraz następująco: $\{(4, 5), (3, 4)\}$. Ruch doprowadza do poprawienia wyniku, zatem *chances* pozostaje bez zmian.

4. Spośród 2 możliwych ruchów najlepszy jest ruch $(1, 2)$. Nie znajduje się on na liście tabu, zatem zostaje wykonany:



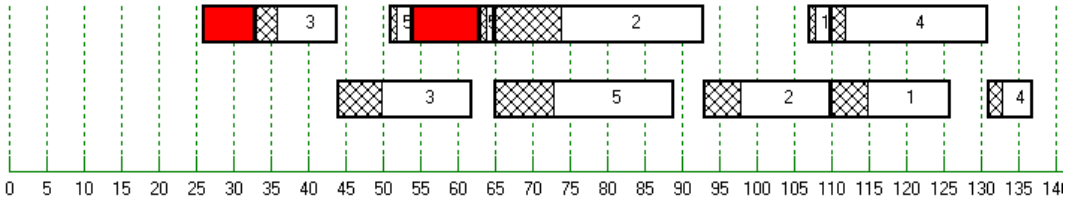
Lista tabu wygląda teraz następująco: $\{(4, 5), (3, 4), (2, 3)\}$. Ruch doprowadza do poprawienia wyniku, zatem *chances* pozostaje bez zmian.

5. Najlepszym ruchem w obecnej sytuacji jest ruch $(2, 3)$. Znajduje się on jednak na liście tabu, a jednocześnie nie spełnia kryterium aspiracji (nie prowadzi do poprawienia najlepszego znalezionego rozwiązania). Jedynym możliwym ruchem jest $(1, 2)$, więc zostaje wykonany:



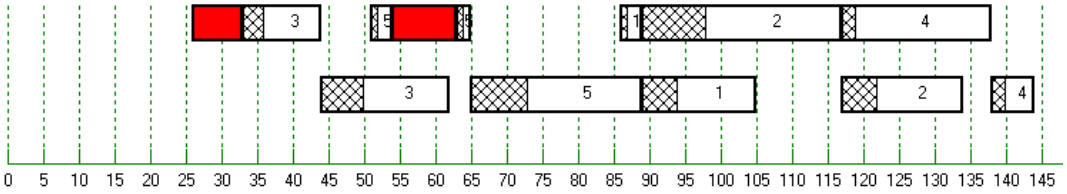
Lista tabu jest pełna, zatem przed dodaniem do niej następnego ruchu usunięty zostaje ruch dodany najwcześniej, czyli $(4, 5)$. Po dodaniu $(1, 2)$ na liście tabu znajdują się następujące ruchy: $\{(3, 4), (2, 3), (1, 2)\}$. Ruch $(1, 2)$ doprowadza do pogorszenia rozwiązania, więc *chances* zostaje obniżone o 1 i wynosi teraz 3.

6. Jedynym możliwym ruchem jest $(4, 5)$, więc zostaje wykonany:



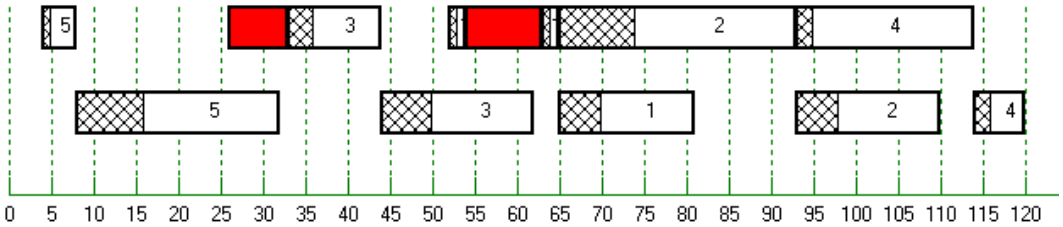
Następuje ponowne pogorszenie wyniku, zatem *chances* zostaje zmniejszone do 2. Na liście tabu: $\{(2, 3), (1, 2), (4, 5)\}$.

7. Po raz kolejny wykonany zostaje jedyny ruch nie znajdujący się na liście tabu, czyli $(3, 4)$:



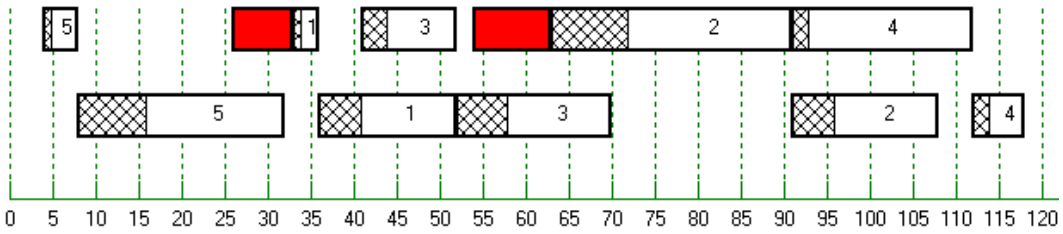
Rozwiązanie ulega dalszemu pogorszeniu: $C_{max} = 144$. Parametr *chances* zostaje obniżony do 1, a lista tabu zaktualizowana: $\{(1, 2), (4, 5), (3, 4)\}$.

8. Przy aktualnym uporządkowaniu najlepszy jest ruch $(1, 2)$. Znajduje się na liście tabu, ale może być wykonany ponieważ spełnia kryterium aspiracji. Prowadzi bowiem do uzyskania rozwiązania o $C_{max} = 120$, które jest lepsze od dotychczasowego najlepszego o wartości 133.



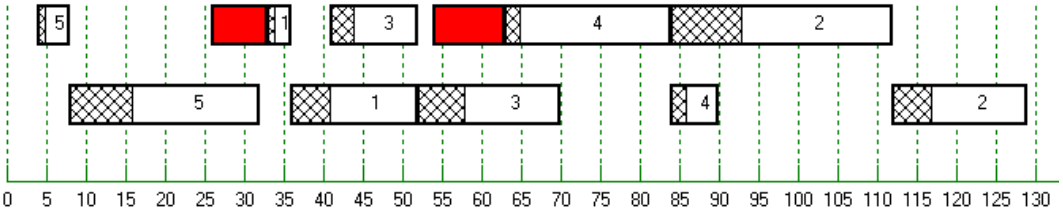
Do listy dodany zostaje wykonany ruch: $\{(4, 5), (3, 4), (1, 2)\}$. Następuje poprawienie rozwiązania, więc *chances* pozostaje bez zmian.

9. Jedyny możliwy ruch $(2, 3)$ okazuje się być najlepszym i zostaje wykonany:



   Do listy dodany zostaje wykonany ruch: $\{(3, 4), (1, 2), (2, 3)\}$. Następuje poprawienie rozwiązania, więc *chances* pozostaje bez zmian.

10. Możliwy jest tylko ruch $(4, 5)$, a żaden z ruchów na liście tabu nie prowadzi do poprawienia najlepszego rozwiązania. Zatem ruch $(4, 5)$ zostaje wykonany:



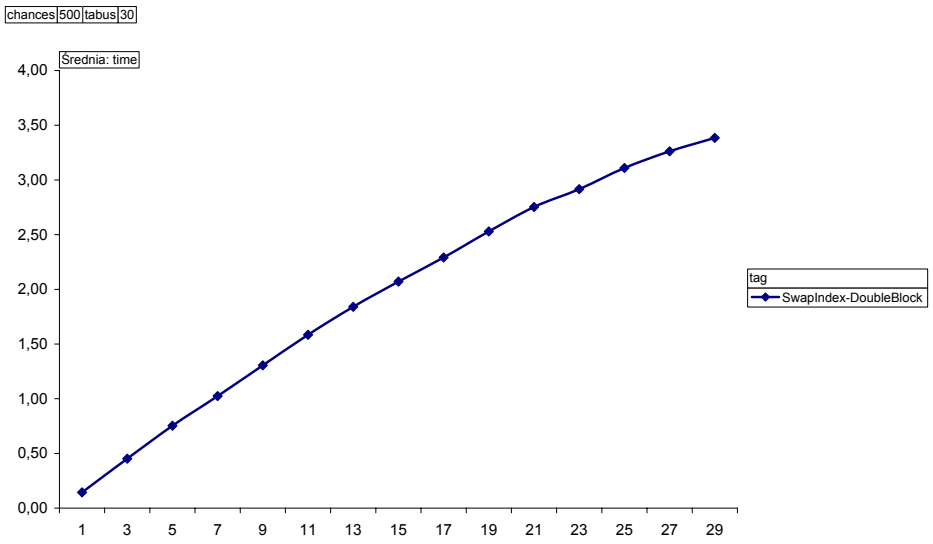
   Ruch ten pogorsza funkcję celu, co powoduje obniżenie parametru *chances* do 0. To z kolei jest warunkiem zakończenia algorytmu. Najlepszym znalezionym rozwiązaniem jest uporządkowanie o wartości $C_{max} = 118$ z kroku 8.

3.3 Optymalizacja parametrów

W celu znalezienia optymalnych parametrów dla algorytmu przeprowadzono serię pomiarów na stosunkowo niewielkich ($n = 50$) instancjach „próbnych”.

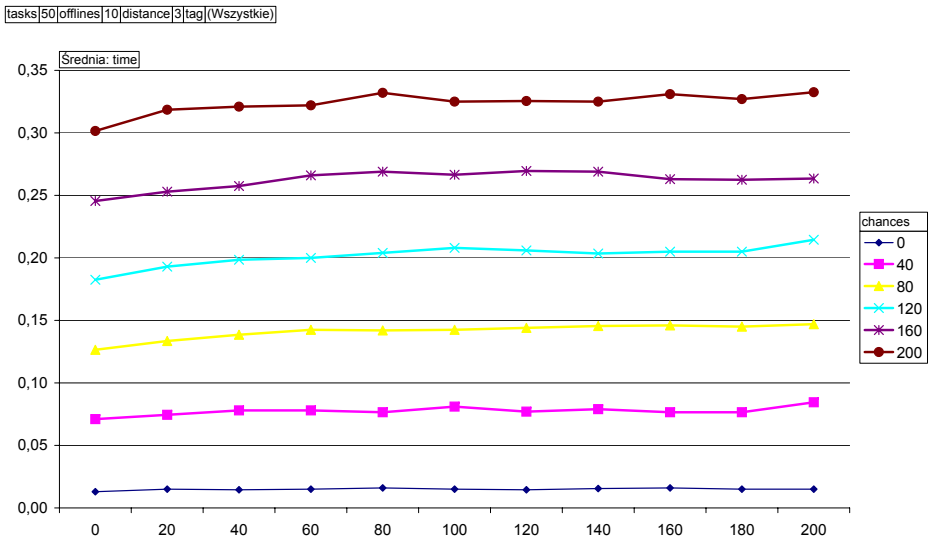
3.3.1 Wpływ parametrów na czas obliczeń

Przed przystąpieniem do optymalizacji parametrów algorytmu konieczne było sprawdzenie jaki wpływ mają one na czas trwania obliczeń, aby w dalszym toku eksperymentu preferować takie zmiany parametrów, które powodują możliwie najmniejsze opóźnienia w otrzymywaniu wyników.



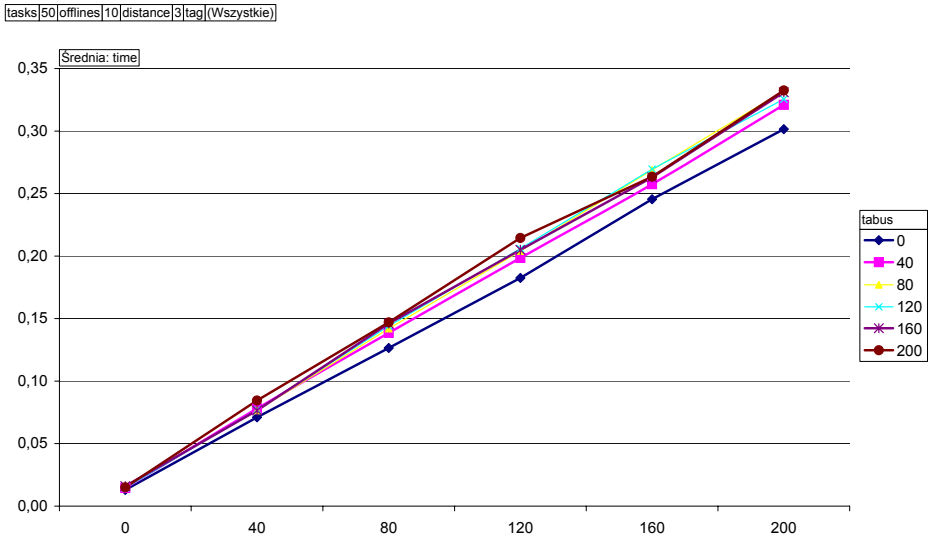
Rysunek 2: Czas obliczeń w zależności od zmieniającego się *distance*

Parametr *distance* Rysunek 2 przedstawia wykres zależności czasu obliczeń od zmieniającej się maksymalnej odległości między zamienianymi zadaniami. Wykres jest zgodny ze wzorem na ilość możliwych ruchów, danym w 2.2.3, i przedstawia zależność zbliżoną do liniowej.



Rysunek 3: Czas obliczeń w zależności od zmieniającego się *tabus*

Parametr *tabus* Badania wpływu długości listy tabu na czas obliczeń potwierdziły teoretyczne przewidywania oparte na analizie kodu. Ze względu na stały koszt $O(1)$ operacji wykonywanych na liście tabu, parametr ten nie wpływa na czas obliczeń. Obrazuje to wykres na rysunku 3.

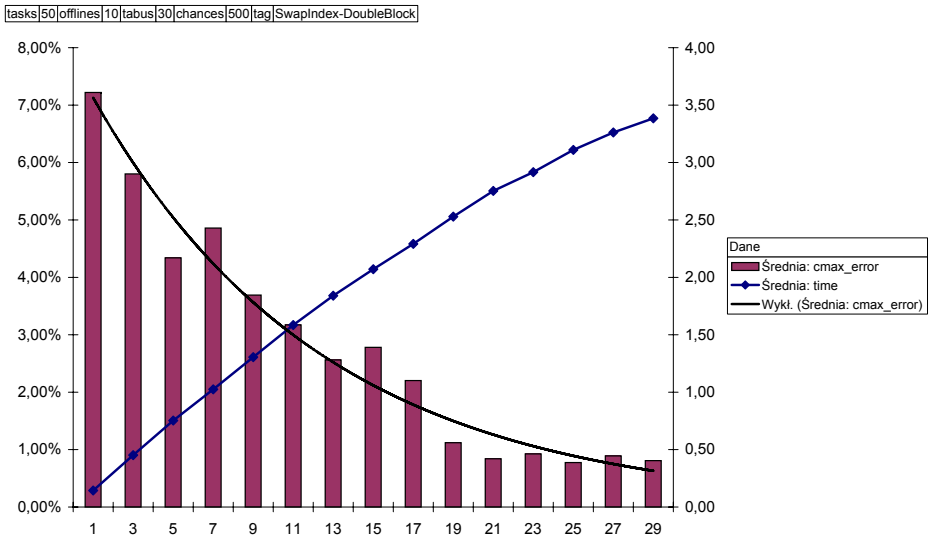


Rysunek 4: Czas obliczeń w zależności od zmieniającego się *chances*

Parametr *chances* Zgodnie z przewidywaniami ilość dodatkowych iteracji ma liniowy wpływ na czas obliczeń. Algorytm wykonuje zawsze $x + chances$ iteracji, gdzie x to sumaryczna ilość iteracji, w których poprawia się wynik. Obrazuje to wykres na rysunku 4.

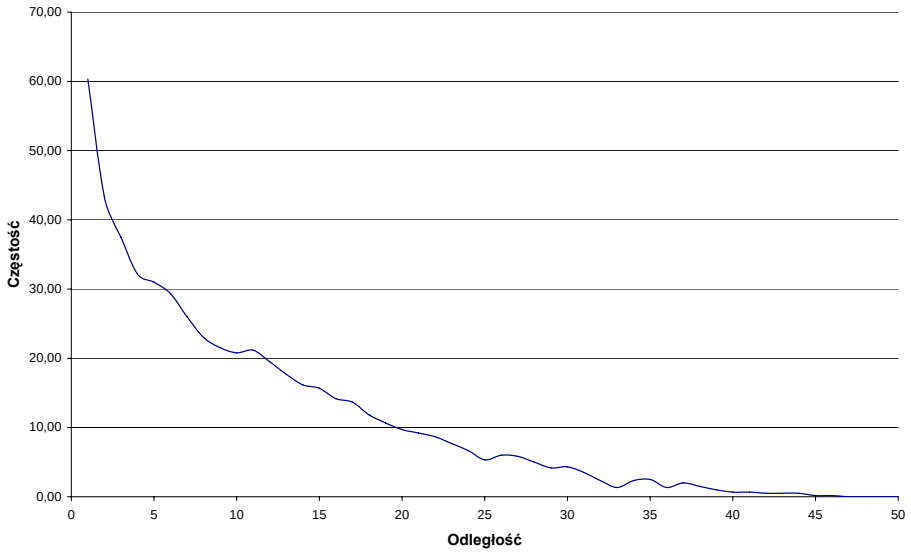
3.3.2 Wpływ parametrów na jakość rozwiązania

Na podstawie wyników pomiarów prędkości działania algorytmu w następnym kroku eksperymentu zdecydowano się przeprowadzić stosunkowo niewielką ilość pomiarów dla parametru *distance* (duży koszt czasowy), większą uwagę poświęcając parametrom *tabus* i *chances*.

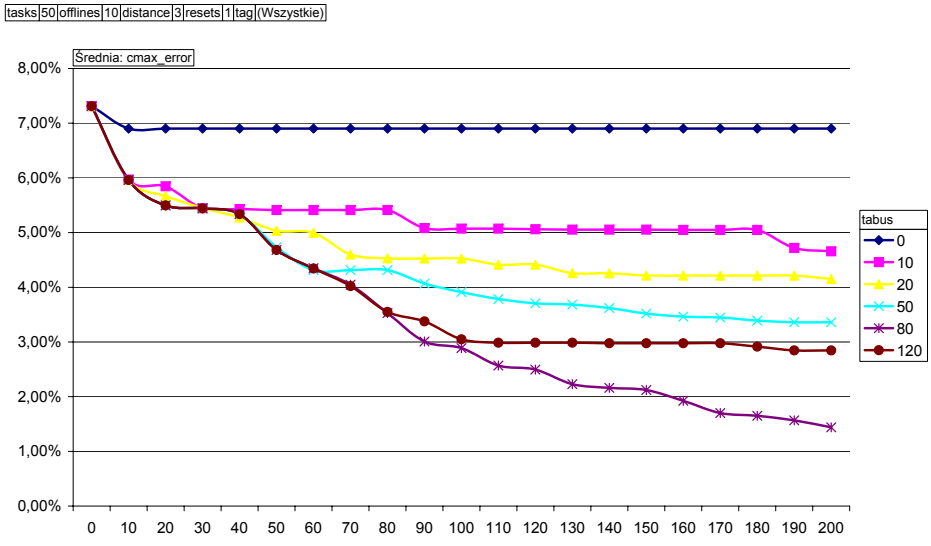


Rysunek 5: Porównanie jakości i prędkości dla zmieniającego się *distance*

Parametr *distance* Rysunek 5 przedstawia wykres zależności jakości rozwiązania i czasu jego obliczania od zmieniającej się maksymalnej odległości między zamienianymi zadaniami. Dobrym przybliżeniem krzywej jakości rozwiązania jest krzywa wykładnicza. Powodem takiego zachowania algorytmu jest to, że w praktyce zamiany bardziej odległych pozycji są wybierane znacznie rzadziej. Fakt ten obrazuje wykres na rysunku 6.



Rysunek 6: Częstość wyboru ruchu w zależności od jego *distance*



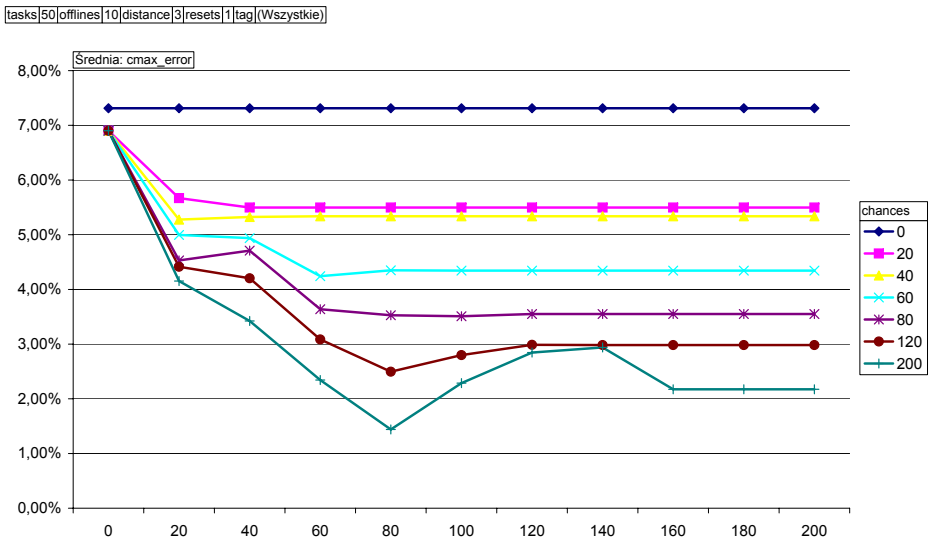
Rysunek 7: Jakość rozwiązania w zależności od zmieniającego się *distance*

Parametr *chances* Rysunek 7 przedstawia wykres zależności jakości rozwiązania od zmieniającej się ilości dodatkowych iteracji.

Analiza wykresu prowadzi do następujących wniosków:

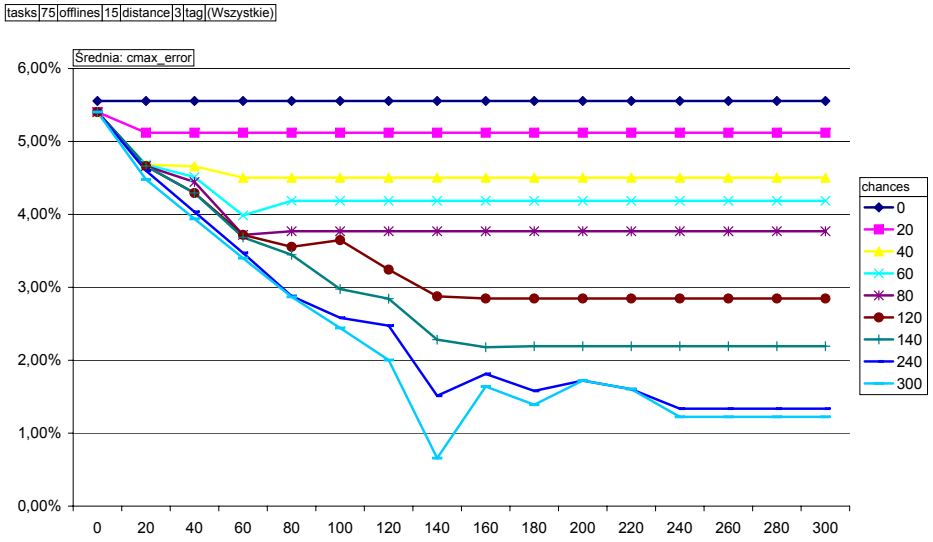
- zwiększanie wartości *chances* daje najlepsze efekty dla *chances* $\leq$ *tabus*,
- dla *chances* $>$ *tabus* może wystąpić dalsza poprawa, ale nie musi, nigdy natomiast nie występuje pogorszenie,
- dla tej samej wartości *chances* (ten sam czas działania) ważne jest odpowiednie dobranie wartości *tabus*,
- wartość *tabus* może być za duża (seria 120) lub za mała (seria 20) i zwiększanie *chances* przynosi niewielką poprawę,
- dla idealnie dobranego *tabus* (seria 80) następuje stała poprawa wyniku.

Powyższe fakty można wytłumaczyć w następujący sposób. Jeżeli *chances* $<$ *tabus*, to lista tabu często nie zdąży się wypełnić przed przerwaniem algorytmu i działa wtedy tak, jakby miała długość równą *chances* a nie *tabus*. Szczególnie widoczne jest to dla dużych wartości *tabus*, np. 50, 80 i 120. W przypadek *tabus* $= 50$ niemożliwe jest dalsze ulepszanie rozwiązania, ponieważ zbyt krótka lista tabu pozwala wrócić do wcześniejszych rozwiązań, natomiast w przypadku *tabus* $= 120$ blokowanych jest zbyt wiele ruchów. Przypadek pośredni, *tabus* $= 80$ daje najlepsze rezultaty.



Rysunek 8: Jakość rozwiązania w zależności od zmieniającego się *tabus*

Parametr *tabus* Rysunek 8 przedstawia wykres zależności jakości rozwiązania od zmieniającej się długości listy tabu. Analiza wykresu prowadzi do wniosków pokrywających się z rozważaniami nad wykresem na rysunku 7. Wyraźnie widoczne jest tutaj, że najlepsza wartość *tabus* ujawnia się dopiero dla *chances* $>$ *tabus*, na wykresie objawia się to „dołkiem" dla serii *chances* > 80. Analiza kolejnego wykresu, przedstawionego na rysunku 9 prowadzi do wniosku, że ów „dołek", czyli obszar o najmniejszym błędzie względnym, występuje dla *tabus* $\approx 30\,\%$ *moves*, gdzie *moves* to ilość możliwych ruchów, określona wzorem z punktu 2.2.3.



Rysunek 9: Jakość rozwiązania w zależności od zmieniającego się *tabus*

3.3.3 Wnioski końcowe

Powyższe rozważania prowadzą do dwóch głównych wniosków:

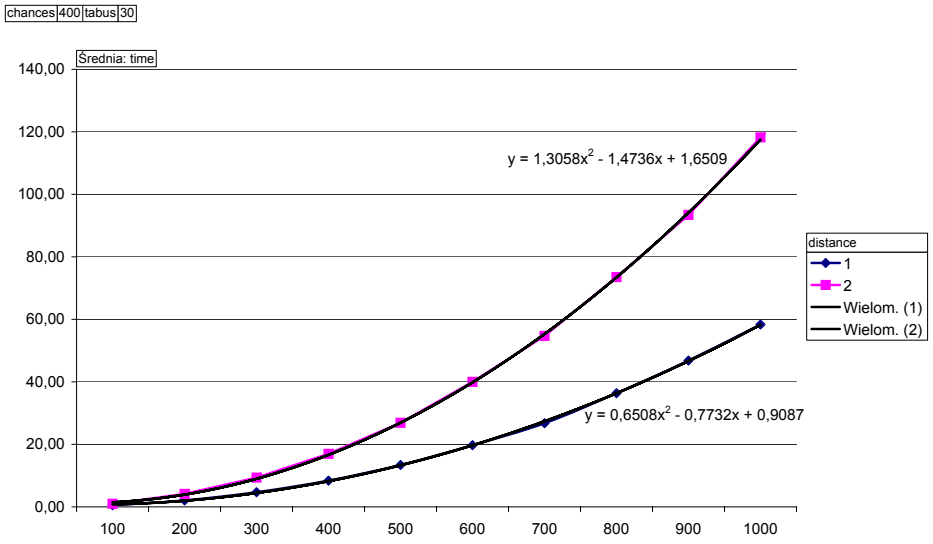
1. Największy wpływ zarówno na jakość jak i czas ma parametr *distance*. W praktyce najlepiej stosować wartości rzędu kilka-kilkanaście, w zależności od oczekiwanej jakości i dostępnego czasu.

2. Najlepsza wartość *tabus* wynosi ok. 30% możliwych ruchów.

3. Najlepsza wartość *chances* jest z pewnością większa od *tabus*, w ogólności im większa, tym lepsza (ale uwaga na czas obliczeń).

3.4 Testy wydajnościowe

Ostatnim z etapów eksperymentu był test wydajnościowy, który miał na celu pokazanie jak algorytm zachowuje się dla różnych liczb zadań przy ustalonych wcześniej parametrach. Wykres na rysunku 10 przedstawia wyniki pomiarów, oraz idealnie dopasowane do nich krzywe wielomianowe. Potwierdza to w sposób ostateczny wyznaczoną złożoność rzędu $O(n^2)$.



Rysunek 10: Czas obliczeń w zależności od liczby zadań