Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

Metody lokalnych popraw w konstrukcji algorytmów rozwiązywania problemu flow-shop

Jakub Lisowski

Praca magisterska
napisana pod kierunkiem
dr Mieczysława Wodeckiego

Wrocław, 2002 r.

Spis treści

1 Wstęp

W pracy zbadano możliwości popraw najlepszych znanych obecnie rozwiązań przykładów Taillarda [3] dla problemu flow-shop zamieszczonych na stronie internetowej
http://www.eivd.ch/ina/collaborateurs/etd/articles.dir/benchmark.ps .
Główne kierunki badań mają na celu:
1. Poprawę algorytmu typu konstrukcyjnego NEH
2. Modyfikację sposobu przeglądania przestrzeni rozwiązań dopuszczalnych w algorytmie opartym na metodzie tabu-search [2].
W sumie zbadano 63 modyfikacje algorytmu oraz poprawiono najlepsze rozwiązania dla 16 przykładów spośród 120.

2 Opis problemu szeregowania typu flow-shop

W problemie przepływowym **flow-shop** mamy zbiór n zadań $J = \{j_1, j_2, \ldots j_n\}$ oraz zbiór m maszyn $M = \{m_1, m_2, \ldots m_m\}$.
Każde zadanie ze zbioru J należy wykonywanć, bez przerywania, na każdej maszynie ze zbioru M w zadanej kolejności, jednakowej dla wszystkich zadań. Z każdym zadaniem jest związany wektor $\vec{C_j} = [c_1, c_2, \ldots c_m]$, czasów wykonywania zadania j odpowiednio na maszynach m_1, m_2 aż do m_m. Czasy te mogą się różnić dla poszczególnych zadań.
Na jednej maszynie w danej chwili może być wykonywane tylko jedno zadanie, podobnie jedno zadanie może być wykonywane w danej chwili tylko na jednej maszynie. Zadania nie mogą być wykonywane równolegle, maszyny są jednozadaniowe. Należy wyznaczyć taką permutację π (kolejność wykonywania zadań) zbioru J, w której czas wykonania wszystkich zadań będzie możliwie najmniejszy.

Problem ten jest oznaczamy w notacji Grahama przez $F|m|C_{max}$, (F, rodzaj problemu - problem permutacyjny, m liczba maszyn, minimalizujemy C_{max}).

W 1976 roku Garey, Johnson i Seti pokazali, że już $F|3|C_{max}$ jest problemem NP-trudnym, więc aby mieć pewność znalezienia rozwiązania optymalnego należy sprawdzić wszystkie n! permutacje zadań.

Jeżeli komputer sprawdza wszystkie możliwe rozwiązania dla 10 zadań w 10 sekund, to dla 20 zadań czas ten rośnie (zakładając stały czas generowania i sprawdzania pojedynczej permutacji) do 212800 lat.

W rozwiązywaniu tego typu problemów stosuje się zazwyczaj algorytmy podziału i ograniczeń (branch & bound), programowania dynamicznego oraz metody gradientów. Dają one co prawda rozwiązanie optymalne, jednak ich czas działania nie jest wielomianowy.
Najlepsze znane algorytmy dokładne oparte na metodzie podziału i ograniczeń rozwiązujące problem flow-shop, napotykają na trudności (czyli ich czas działania przestaje być akceptowalny) przy 20 zadaniach i 5 maszynach.
Z drugiej strony stosuje się także algorytmy dające rozwiązania przybliżone, korzystające z heurystyk: tabu search, symulowango wyżarzania, algorytmów genetycznych lub kolejki priorytetowej.

3 Kryterium optymalizacyjne

W rozwiązywanym probelmie mależy wyznaczyć kolejność wykonywania zadań, która zminimalizuje czasu wykonywania się wszystkich zadań na wszystkich maszynach.

W celu ułatwienia obliczenia wartości funkcji celu wprowadzimy reprezentację grafową problemu: acykliczny graf skierowany, w którym wierzchołki tworza kratę i symbolizują wykonywanie się zadania na pojedynczej maszynie, natomiast łuki zależności technologiczne i czasowe.
W tej reprezentacji każda kolumna wierzchołków odpowiada jednemu zadaniu, a wiersz odpowiada jednej maszynie. Łuki przebiegające pionowo w dół symbolizują porządek technologiczny i kolejność maszyn dla danego zadania. Natomiast, łuki poziome z lewej na prawą stronę, symbolizują uporządkowanie zadań.
Z każdym wierzchołkiem (określanym jako $v_{\pi(i),j}$, gdzie $\pi(i)$ oznacza i-te zadanie w rozpatrywanym uporządkowaniu zadań π, a j oznacza j-tą maszynę) jest powiązana waga $C_{\pi(i)}[j]$, czyli czas wykonywania zadania $\pi(i)$ na maszynie j-tej.

Czas wykonania się wszystkich zadań jest równy $\sum_{v_{\pi(i),j}} C_{\pi(i)}[j]$, gdzie $v_{\pi(i),j}$ należy do najdłuższej (najbardziej obciążonej wagami wierzchołków)

drogi łączącej wierzchołek $v_{\pi(1),1}$ z wierzchołkiem $v_{\pi(n),m}$ biegnącej zgodnie z kierunkami łuków.

Czas ten jest równy:

$$\mathbf{C_{max}}(\pi) = \max_{1 \leq t_1 \leq t_2 \leq \ldots \leq t_{m-1} \leq m} \left(\sum_{j=1}^{t_1} C_{\pi(j)}[1] + \sum_{j=t_1}^{t_2} C_{\pi(j)}[2] + \ldots + \sum_{j=t_{m-1}}^{m} C_{\pi(j)}[m] \right).$$

Aby go obliczyć, do każdego wierzchołka w grafie dodajemy wagę $T_{\pi(i),j}$ oznaczającą czas zakończenia wykonywania się zadania $\pi(i)$ na maszynie j-tej. Wielkości te można obliczyć według rekurencyjnego schematu:

$$\begin{cases} T_{\pi(1),1} = C_{\pi(1)}[1], & \\ T_{\pi(1),j} = C_{\pi(1)}[j] + T_{\pi(i),j-1}, & j > 1, \\ T_{\pi(i),1} = C_{\pi(i)}[1] + T_{\pi(i-1),1}, & i > 1, \\ T_{\pi(i),j} = max(T_{\pi(i-1),j}, T_{\pi(i),j-1}) + C_{\pi(i)}[j], & i > 1 \wedge j > 1. \end{cases}$$

Przy czym $C_{max}(\pi) = T_{\pi(n),m}$.
Złożoność obliczeniowa wyznaczenia tych wielkości jest równa $O(m * n)$.
Najdłuższa droga w grafie (o długości $c_{max}(\pi)$) nazywana jest **ścieżką krytyczną**.

4 Technika lokalnych popraw

W metodzie lokalnych popraw startując z pewnego rozwiązani początkowego wprowadzamy do niego nieznaczne zmiany usiłując je poprawić (czyli w tym przypadku zmniejszyć czas wykonywania wszystkich zadań).

Jako rozwiązanie początkowe możemy przyjąć dowolne uporządkowanie zadań, jednak wartość funkcji celu, dla tego ustawienia może być duża. Korzystne jest, aby przyjęte rozwiązanie początkowe miało możliwie małą wartość funkcji celu. Aby to osiągnąć, stosuje się algorytmy konstrukcyjne wyznaczające rozwiązanie początkowe o możliwie najlepszej (najmniejszej) funkcji celu. Takim algorytmem jest algorytm **NEH** o złożoności obliczeniowej $O(n^2 * m)$ wyznaczający, jak zostanie pokazane w rozdziale 11 *,,Wyniki obliczeniowe"*, rozwiązania oddalone o około 1% od najlepszych znanych rozwiązań. Stosując następnie algorytm popraw możemy oczekiwać, że otrzymamy ,,bardzo dobre" rozwiązanie.

Złożonośc obliczeniowa całego algorytmu lokalnych popraw może wynosić $O(I*S*n*m)$, gdzie I jest liczbą iteracji algorytmu a S liczbą sprawdzanych permutacji podczas jednej iteracji. Iloczyn $n*m$ to koszt obliczania funkcji celu dla danego ustawienia, a pojedyncze ustawienie generowane jest w czasie stałym. Dlatego zastosowanie algorytmu konstrukcyjnego dla generowania rozwiązania początkowego nie zwiększa złożoności obliczeniowej algorytmu popraw, a daje dobry punkt startowy o małej wartości funkcji celu.
Metody lokalnych popraw mają dodatkowo tę zaletę, że parametr I - ilość iteracji można dowolnie zwiekszyć, co pozwala oczekiwać lepszych wyników.

W programach napisanych na potrzeby niniejsej pracy zostanie zastosowany zmodyfikowany algorytm NEH, o którym w dalszej części pracy.

5 Sposoby poprawiania rozwiązania

Szukamy lepszego rozwiązania od startowego przez przestawianie pewnych zadań w permutacji π zbioru J.
W celu zmniejszenia ilości niecelowych (to znaczy nie dających poprawy rozwiązania) przestawień skorzystamy z własności bloków na ścieżce krytycznej. Dokładnie jest to opisane w pracy [1].
W reprezentacji grafowej problemu przedstawionej w poprzednim rozdziale *ścieżka krytyczna* ma postać schodów idących od lewego górnego wierzchołka kraty do prawego dolnego jej wierzchołka, zgodnie z kierunkami łuków.
Blokiem na ścieżce krytycznej nazywamy pojedynczy poziomy ,,schodek" zawierający przynajmniej dwa zadania.

Lemat 1:
Przestawianie zadań wewnątrz dowolnego bloku nie prowadzi do rozwiązania o malejącej wartości funkcji celu.
Dowód:
Załóżmy, że istnieje ścieżka krytyczna S, na której występują bloki $B_1, B_2, \ldots B_k$. Wagą ścieżki S jest $\sum_{i=1}^{k} B_i$.
Blok jest ciągiem wierzchołków $B_i = [b_1, b_2, \ldots b_b]$.Tak więc za wagę bloku można przyjąć $\sum_{j=1}^{b} b_j$.
Jeżeli zamienimy wewnątrz bloku wierzchołki b_i i b_j, $i, j \in \{1 \ldots b\}, i > 1$, $j < b$, (co oznacza także zamianę odpowiednich kolumn wierzchołków

w całym grafie), to waga bloku się nie zmieni (z przemienności dodawania), podobnie jak waga pozostałych bloków na ścieżce S. Nową ścieżkę z zamienionymi wierzchołkami oznaczymy przez S'. Jeżeli zamiana kolumn nie spowodowała wygenerowania nowej ścieżki krytycznej, to ścieżka S' mająca taką samą wagę jak ścieżka S w pierwotnym grafie jest nadal ścieżką krytyczną (czyli wartość rozwiązania nie uległa zmianie).
Jeżeli natomiast, powstała nowa ścieżka krytyczna S'' o wadze większej niż waga ścieżki S', to nowo wygenerowana permutacja jest gorszym rozwiązaniem.

□

Wiele innych własności bloków ścieżki krytycznej jest przedstawionych między innymi w pracy [1].

6 Ocena jakości algorytmu

Przedstawione w tej pracy algorytmy heurystyczne rozwiązywania problemu szeregowania zadań typu flow-shop były testowane na przykładach Taillarda [3] zamieszczonych na stronie internetowej pod adresem:
http://www.eivd.ch/ina/collaborateurs/etd/articles.dir/benchmark.ps .
Jest to reprezentatywny zestaw danych zawierający 120 przykładów, na którym testowane są obecnie wszystkie algorytmy rozwiązywania rozpatrywanego problemu. Oprócz danych znajdują się tam także najlepsze obecnie znane dolne i górne oszacowania optymalnych rozwiązań. Dokładne rozwiązania w ogromnej części nie są znane.
Dla oceny rozwiązania wyznaczonego przez algorytm stosuje się:
a) różnicę wartości (odległość):

$$O_{dolne} = C_{algorytmu} - C_{dolne}$$

b) względną odległość:

$$B_{dolne} = \frac{C_{algorytmu} - C_{dolne}}{C_{dolne}}.$$

Drugim współczynnikiem jakości algorytmu jest różnica pomiędzy wartością wyznaczonego rozwiązania, a górnym ograniczeniem, czyli:

$$O_{górne} = C_{algorytmu} - C_{górne}, \qquad B_{górne} = \frac{C_{algorytmu} - C_{górne}}{C_{górne}}.$$

Średnie wartości powyższych współczynników, dla przedstawionych w pracy algorytmów, są zamieszczone w rozdziałach, w których opisywane są poszczególne algorytmy.

7 Algorytm heurystyczny NEH

Oryginalny algorytm NEH został przedstawiony przez Nawaz, Enscore i Ham (korzystałem z zapisu algorytmu zamieszczonego w [2]) w 1988 roku. Wyznaczone przez ten algorytm rozwiązania są punktami startowymi wielu algorytmów typu popraw.

Algorytm NEH:

Krok 1: Uporządkuj malejąco zadania względem całkowitego czasu wykonywania (sumy czasów na wszystkich maszynach).

Krok 2: **k=2**, wybierz dwa pierwsze zadania i ustaw je tak, aby zminimalizować czas ich całkowitego (czas wykonania ostatniego zadania na ostatniej maszynie) wykoniania. Przyjmij lepsze ustawienie za bieżące rozwiązanie.

Krok 3: Zwiększ **k** o 1. Weź **k**-te zadanie, wstawiając je kolejno pomiędzy zadania z bieżącego rozwiązania znajdź najlepsze uporządkowanie **k** zadań, przyjmij je za bieżące rozwiązanie.

Krok 4: Jeżeli **k**=**n**, zakończ pracę, w przeciwnym wypadku przejdź do kroku 3.

Złożoność obliczeniowa sortowania w kroku pierwszym wynosi $O(n*log(n))$. Krok 2 wykonuje się w czasie $O(m)$ - koszt obliczenia czasu dla jednego ustawienia w kroku 3, wynosi $O(k * m)$, $(k = 3 \ldots n)$. Jednak jest możliwe obliczenie całego kroku 3 (dla wszystkich wartości i od początku tablicy do k) w czasie $O(km)$, co daje złożoność obliczeniową algorytmu NEH $O(n^2 * m)$.

Całkowity czas wykonywania C_i (czas po wstawieniu zadania **k**-tego na i-tą pozycję w permutacji π) można wyznaczyć następująco:

1. Obliczyć czasy $T_{\pi(i),j}$ zakończenia zadania i-tego na maszynie j-tej w ustawieniu π (uporządkowanie dla pierwszych k-1 zadań), według wzoru:

   - $T_{\pi(0),j} = 0$,
   - $T_{\pi(i),0} = 0$,
   - $T_{\pi(i),j} = max(T_{\pi(i),j-1}, T_{\pi(i-1),j}) + c_{\pi(i)}[j]$,

   $(i = 1, 2, \ldots, k-1), (j = 1, 2, \ldots, m)$.

2. Obliczyć $q_{\pi(i),j}$, czasy pomiędzy rozpoczęciem wykonywania zadania $\pi(i)$-tego na j-tej maszynie, a całkowitym czasem wykonywania zadań $C_{max}(\pi)$:

   - $q_{k,j} = 0$,
   - $q_{i,m+1} = 0$,
   - $q_{i,j} = max(q_{i,j+1}, q_{i+1,j}) + c_{\pi(i)}[j]$,

   $(i = k-1, \ldots, 1), (j = m, \ldots, 1)$.

3. Obliczyć najmniejszy czas $f_{\pi(i),j}$ na j-tej maszynie, dla zadania k-tego wstawionego na pozycję i:

   - $f_{i,0} = 0$,
   - $f_{i,j} = max(f_{i,j-1}, T_{\pi(i-1),j}) + c_k[j]$,

   $(i = 1, 2, \ldots, k), (j = 1, 2, \ldots, m)$.

4. Wartość $C_{max}(\pi')$, gdzie permutacja π' powstaje z permutacji π przez wstawienie zadania k-tego na i-tą pozycję:

   $$C_{max}(\pi') = \max_j(f_{i,j}, q_{i,j}),$$

   $(i = 1, 2, \ldots, k), (j = 1, 2, \ldots, m)$.

Wszystkie powyższe kroki można wykonać w czasie $O(k * m)$, dla $(k = 3, \ldots, n)$, co daje złożoność czasową algorytmu $O(n^2 * m)$.

Obecnie algortym NEH uchodzi za najlepszy algorytm typu konstrukcyjnego dla rozwiązywania problemu typu flow-shop.

8 Zmodyfikowany algorytm NEH

W algorytmie NEH, w pierwszym kroku sortuje się zadania według całkowitego czasu ich wykonywania. Jednak problem pojawia się, gdy są dwa lub więcej zadania o takim samym całkowitym czasie wykonywania. Ich wzajemne położenie po posortowaniu względem klucza, który zadania mają identyczny, często ma znaczenie dla końcowego rozwiązania (kolejność ta determinuje czas rozpatrywania zadań, a więc w części przypadków ich kolejność w końcowym rozwiązaniu).

Próba zastosowania najdłuższego czasu wykonywania na pojedynczej maszynie, jako drugiego podczas porównywania zadań, zarówno przesuwając do przodu zadanie z najdłuższym czasem wykonywania na pojedynczej maszynie jak i do tyłu spowodowała pogorszenie się wynikow o około 4 dziesiąte procenta. Jednak minimalnie lepsze okazało się wybieranie jako wcześniejsze zadanie mające dłuższy najdłuższy czas na pojedynczej maszynie. Jako trzeci klucz używany był numer zadania, czyniąc to sortowanie stabilnym.
Szczegółowe wyniki testu i porównanie z innymi wersjami algorytmu NEH są zamieszczone w rozdziale 11 *„Wyniki obliczeniowe”*.

W związku z występowaniem w przykładzie podzbiorów zadań o takim samym całkowitym czasie wykonywania, które podczas sortowania mogą zostać różnie ustawione (co wpływa na wynik algorytmu), aby otrzymać najlepsze rozwiązanie możliwe do wyznaczenia przez algorytm NEH, należy powtórzyć cały algorytm $I = \Pi_{i=1}^{k} z_i!$ gdzie k to ilość zbiorów zadań o takim samym całkowitym czasie wykonywania, a z_i to liczność i-tego takiego zbioru. Złożoność tego algorytmu wzrasta w takim przypadku do $O((\Pi_{i=1}^{k} z_i!) * (n^2 * m))$.

Wśród danych testowych są duże zestawy 500 zadań z ponad 100 takimi zbiorami. Jeżeli każdy z tych zbiorów miałby tylko po 2 elementy, cały trwający $O(n^2 * m)$ algorytm trzeba by powtórzyć 2^{100} razy, co jest nie do przyjęcia ze względów czaswych.
Rozwiązaniem jest wprowadzenie ograniczenia ilości rozpatrywanych permutacji występujących po kroku 1 algorytmu lub ilość permutowanych zbiorów zadań o takim samym całkowitym czasie wykonywania.
Jako ograniczenie przyjęto pierwszy warunek.

Zmodyfikowano algorytm NEH tak, aby dla każdego zbioru zadań, sprawdzając od początku posortowaną nierosnąco tablicę zadań, generował jego wszystkie permutacje i obliczał dla nich wynik algorytmu (najpierw generował wszystkie permutacje pierwszego zbioru, następnie generował kolejną permutację drugiego zbioru i znowu wszystkie permutacje pierwszego, następnie kolejną drugiego itd, dla każdego tak wygenerowanego ustawienia przeprowadzał następnie kroki 2-4 algorytmu NEH). Jako dodatkowe ograniczenie czasowe przyjęto 1000 iteracji. W wyniku takiego podejścia czas działania algorytmu wzrasta do 1000 razy (co dla procesora CeleronII taktowanego zegarem 800 MHz może oznaczać czas rzędu 20 sekund dla 200 zadań i 3 minut w przypadku 500 zadań i 500 iteracji algorytmu). Nie gwarantuje nam to możliwie najlepszego wyniku, który można osiągnąć przy szczęśliwym zrandomizowanym sortowaniu w pierwszym kroku algorytmu NEH. Jednak dość znacznie zwiększa to prawdopodobieństwo znalezienia lepszego rozwiązania niż wyznaczanie (podobną ilość razy) rozwiązania standardowym algorytmem NEH.
Podczas sortowania duże znaczenie ma funkcja sortująca. Jeżeli jest ona ,,sztywna", algorytm zachowuje się deterministycznie i wraz ze zwiększeniem ilości iteracji polepsza swe wyniki. Jeżeli natomiast jako funkcję sortującą weźmiemy funkcję porównującą tylko całkowite czasy wykonywania się zadań, oraz przestrzeń możliwych sortowań zadań jest większa niż 1000 elementów (wystarczy co najmniej 10 par elementów o takim samym całkowitym czasie wykonywania), wówczas wyniki mogą się różnić. Jednak jest dość prawdopodobne, że będą lepsze niż w przypadku deterministycznej funkcji sortującej. W trakcie obliczeń zdarzało się otrzymać lepsze rozwiązanie przy mniejszej liczbie iteracji algorytmu.

Należy więc znaleźć kompromis pomiędzy czasem przeznaczonym na znalezienie rozwiązania początkowego (w przypadku opisanej tu modyfikacji algorytmu NEH), a czasem przeznaczonym na działanie samego algorytmu popraw. Chęć znalezienia ostatecznego i najlepszego z możliwych rozwiązań, które może wyznaczyć algorytm NEH, czyli sprawdzenia całej przestrzeni możliwych sortowań w pierwszym kroku algorytmu, może, i w przypadku dużych danych (duża ilość zadań) zazwyczaj będzie prowadzić do wykładniczego czasu działania algorytmu.

Do dalszych obliczeń zamiast ograniczenia 1000 iteracji algorytmu NEH wybrałem 500 iteracji oraz funkcję sortującą porównujacą tylko całkowite

czasy wykonywania zadań. W dlaszej części pracy algorytm ten będzie oznaczony przez **mNEH**.

9 Techniki lokalnych popraw

W rozdziale tym przedstawimy podstawowe elementy metody tabu-search powszechnie obecnie stosowanej do rozwiązywania trudnych problemów kombinatorycznych.

9.1 ,,Najlepszy z bloków"

Dla permutacji startowej π wyznaczymy droge krytyczną oraz bloki na niej leżące. Następnie, dla każdego bloku i dla kazdego zadania w bloku przestawiamy zadanie ,,przed" i ,,za" blok.
Każde przestawienie zadania $\pi(z)$,,przed" w permutacji π dla bloku zaczynającego się na $\pi(i)$-tym zadaniu a kończącego na $\pi(j)$-tym $(i < j)$, gdzie $i < z \leq j$ jest przesunięciem typu *shift* względem ustawienia początkowego zadań w π od miejsca i do $z-1$ o 1 miejsce do przodu oraz wstawienie zadania $\pi(z)$ na miejsce i. Daje ono permutację π', gdzie $\pi'(q) = \pi(q)$ dla $q > 0, q < i \wedge q > z, q < n$, oraz $\pi'(i) = \pi(z)$ i $\pi'(r) = \pi(r+1)$ dla $r \in \{i, \ldots z-1\}$.

Analogicznie przestawienie ,,za" w permutacji π jest przesunięciem zadań od $\pi(z+1)$ do $\pi(j)$ o 1 miejsce do tyłu i wstawienie zadania $\pi(z)$ na miejsce j.
Daje ono permutację π', gdzie $\pi'(q) = \pi(q)$ dla $q > 0, q < z \wedge q > j, q < n$, oraz $\pi'(j) = \pi(z)$ i $\pi'(r) = \pi(r+1)$ dla $r \in \{z+1, \ldots j\}$ w permutacji. Dla każdego tak wyznaczonego przestawienia π' wybiera się najlepsze w danej iteracji i jest ono przyjmowane jako rozwiązanie początkowe π w następnej iteracji.
Nie ma potrzeby przestawiania zadań z bloku zaczynającego się na pierwszej maszynie ,,przed" ten blok, ponieważ jest to przestawienie zadań wewnątrz bloku (wartość ścieżki krytycznej na pewno nie zmaleje). Podobnie, nie ma również potrzeby przestawiania ,,za" blok zadań z bloku kończącego się na ostatniej maszynie.

W każdej iteracji sprawdza się k ($n < k < 2n$) perutacji π', i wybiera najlepszą ze znalezionych (nawet jeżeli wartość funkcji celu jest gorsza, czyli większa od poczatkowej). Aby uniknąc wpadnięcia w cykl, stosuję listę tabu na której umieszczamy pary przesuwanych elementów (element $\pi(z)$ oraz $\pi(i)$ lub $\pi(j)$, zależnie od kierunku przesunięcia).
Najlepsze dotychczas znalezione uszeregowanie zadań pamiętane jest w osobnej permutacji. Jeżeli w danej iteracji zostało znalezione lepsze rozwiązanie, jest podstawiane za najlepsze.

Wadą takiego podejścia jest jedynie niewielki obszar sprawdzonych rozwiązań z przestrzeni wszystkich rozwiązań dopuszczalnych. W każdym kroku iteracji możemy oddalić się od rozwiązania początkowego tylko o niewielką zmianę ustawienia, oraz przy długiej liście tabu względem ilości iteracji, nowe przeglądane rozwiazania mogą nie prowadzić do lepszych.

9.2 Dostrajanie algorytmu

Dość istotną kwestią, w przypadku wyżej opisanego postępowania, jest wybór długości listy tabu. Ze względu na charakter przestawień i pamiętane na liście informacje (nie pamięta się całych ustawień, a tylko przestawiane pary) lista tabu może blokować utworzenie bloku o krótkim czasie przejścia (był taki blok i został zmieniony, a następnie sąsiedni blok uległ poprawie). Zbyt krótka lista może szybko prowadzić do wpadnięcia w cykl i bezproduktywne działanie algorytmu. Jako wyznacznik długości listy tabu możemy przyjąć ilość bloków. Maksymalnie jest ich tyle, ile maszyn.
Sprawdzono zachowanie się algorytmu dla długości listy tabu równej $x * n$, gdzie $x = 1, 2, 3, 4, 5, 6$ (n jest ilością zadań), oraz dla ustalonej długości listy tabu długości 5,6,7,8,9,10,11,12,13,14,15,20,25,50.
Ze względu na stosowanie, przy wyznaczaniu rozwiązania początkowego w algorytmie mNEH zrandomizowanej funkcji Quicksort, pomimo 500 iteracji wyniki mogą być różne (różne uporządkowania danych dla tych samych lub różnych wartości C_{mNEH}) dla programów kompilowanych różnymi kompilatorami oraz różnych uruchomień algorytmu. Z tego względy wykonano po kilka testów. Testy były wykonywane oddzielnie, aby ustalić, czy lepsza jest stała czy też zmienna lista tabu.
Ilość iteracji algorytmu wynosiła maksimum z 500 lub $n * log(n)$, co zapew-

nia odpowiednią liczbę iteracji dla zestawów z niewielką ilością zadań, oraz odpowiednią ilość iteracji dla zestawów dużych, gdzie przestrzeń potencjalnych rozwiązań rosła wykładniczo.

W pracy [1] autorzy wykorzystywali listę tabu o długości $6 + \lfloor(\frac{n}{10m})\rfloor$. Jako dodatkowy test umieściłem ją także w tabeli testów dla listy tabu o stałej długości, jako program o identyfiaktorze **6xe**.

Tabela 1: Wyniki testów dla zmiennej długości listy tabu:

Identyfikator programu	$\mathbf{O_{dolne}}$	$\mathbf{B_{dolne}}$	$\mathbf{O_{górne}}$	$\mathbf{B_{górne}}$
1m$^{(*)}$	194.35833	**0.05**54156989	33.59166	**0.00**77708465
	195.1666	**0.05**55317949	34.4	**0.00**78894114
2m	198.74166	**0.05**68093107	37.975	**0.00**90123275
	198.866	**0.05**68921257	38.1	**0.00**90972537
3m	200.075	**0.05**75094916	39.30833	**0.00**96671835
	200.825	**0.05**76866971	40.05833	**0.00**98416790
4m	203.94166	**0.05**89425194	43.175	**0.01**09479837
	204.225	**0.05**90082608	43.45833	**0.01**10140056
5m	205.633	**0.05**91883867	44.866	**0.01**11833397
	204.975	**0.05**91878089	44.20833	**0.01**11844650
6m	204.94166	**0.05**92522661	44.175	**0.01**12501859
	208.025	**0.05**95346441	47.25833	**0.01**15252449
Średnio	201,6478258333	**0,05**79132504	40,8811925	**0,01**00319938

* **t**m oznacza algorytm z długością listy tabu równą $t * m$.

Tabela 2: Wyniki testów dla listy tabu o stałej długości.

Identyfikator programu	O_{dolne}	B_{dolne}	$O_{górne}$	$B_{górne}$
5e$^{(*)}$	205.625	**0.05**67458240	44.85833	**0.00**90435279
6e	197.3166	**0.05**59929892	36.55	**0.00**83341156
7e	194.6833	**0.05**49289014	33.9166	**0.00**73670298
	193.44166	**0.05**47228463	32.675	**0.00**71627214
8e	193.025	**0.05**48409220	32.25833	**0.00**72802232
9e	196.85	**0.05**53786859	36.0833	**0.00**77714269
10e	194.2	**0.05**51460669	33.433	**0.00**75594952
	196.275	**0.05**51733805	35.50833	**0.00**75840152
11e	192.75833	**0.05**47642524	31.99166	**0.00**71906256
	193.7166	**0.05**47863553	32.95	**0.00**72119813
12e	195.15833	**0.05**552390	34.39166	**0.00**78764802
	193.1	**0.05**53866067	32.333	**0.00**77414384
13e	194.65	**0.05**528271301	33.8833	**0.00**76640984
	194.59166	**0.05**53422713	33.825	**0.00**77220741
14e	194.0166	**0.05**52924103	33.25	**0.00**76601436
	194.975	**0.05**52886349	34.20833	**0.00**76579924
15e	193.54166	**0.05**57639329	32.775	**0.00**81063802
	196.25	**0.05**60444340	35.4833	**0.00**83841206
20e	196.99166	**0.05**61086272	36.225	**0.00**84224106
	195.4	**0.05**59707860	34.633	**0.00**82817906
25e	198.375	**0.05**65334220	37.60833	**0.00**87971249
	197.75	**0.05**64243924	36.9833	**0.00**86901393
50e	202.49166	**0.05**83152591	41.725	**0.01**04274119
	200.675	**0.05**82389398	39.908333	**0.01**03510449
6xe	195.625	**0.05**58544592	34.858333	**0.00**81966722
Średnio	196.0593224	**0.05**57540405	35.29261744	**0.00**80993793

* **t**e oznacza algorytm z długością listy tabu równą t.

Dla pewnych długości listy tabu przeprowadzono po dwa testy, gdyż pomimo zastosowania zmodyfikowanego algorytmu NEH, jego wyniki nadal mogą być różne dla róznych kompilacji i uruchomień (mogą być różne wartości funkcji C_{NEH} oraz ustawienia zadań), co ma znaczący acz trudno badalny

wpływ na rozwiązanie wyznaczane przez algorytm.

Najlepsze wyniki algorytmu otrzymano dla listy tabu o stałej długości równej 11 (najmniejsze odchylenie bezwzgledne od dolnych ograniczeń), oraz 7 (ze względu na druga najmniejszą wartość odchylenia względnego).

9.2.1 Ostateczna wersja algorytmu typu ,,najlepszy z bloków"

W celu sprawdzenia wpływu skrócenia czasu działania algorytmu (zarówno podczas wyznaczania rozwiązania poczatkowego jak i podczas działania algorytmu popraw) na otrzymane wyniki przetestowano następujące wersje algorytmu (każda wersja miała listę tabu długości 11 elementów):

1. rozwiązanie początkowe wyznaczono algorytmem mNEH, a algorytm tabu search wykonywał $max(500, n * log_e n)$ iteracji

2. algorytm tabu wykonywał 500 iteracji, a rozwiązanie początkowe wyznaczono algorytmem mNEH

3. algorytm tabu wykonywał 500 iteracji. Jako rozwiązanie początkowe wykorzystywał wynik algorytmu NEH, w którym podczas pierwszego kroku (sortowania) użyta była funkcja sortująca sprawdzająca łączny czas wykonywania się zadania jako główne kryterium, a najdłuższy czas wykonywanie zadnia na którejkolwiek maszynie jako kryterium drugie. Natomiast jako trzecie numer zadania .

4. algorytm wykonywał $max(500, n * log_e n)$ iteracji i jako rozwiązanie początkowe wykorzystywał wynik algorytmu NEH, podobnie jak w przypadku 2.

Tabela 3: Wyniki testów poszczególnych wersji algorytmu.

Grupa	Prog. 1	Prog. 2	Prog. 3	Prog. 4
20 x 5	**0.02**872482[1)] 34.7[2)]	**0.02**872482 34.7	**0.02**872482 34.7	**0.02**872482 34.7
20 x 10	**0.10**293204 142.7	**0.10**293204 142.7	**0.10**309688 142.8	**0.10**309688 142.8
20 x 20	**0.22**040979 407.9	**0.22**040979 407.9	**0.22**040979 407.9	**0.22**040979 407.9
50 x 5	**0.01**031568 28.1	**0.01**031568 28.1	**0.01**031568 28.1	**0.01**031568 28.1
50 x 10	**0.03**520095 102.6	**0.03**520095 102.6	**0.03**714256 108.1	**0.03**714256 108.1
50 x 20	**0.12**176355 414.5	**0.12**176355 414.5	**0.12**247952 416.8	**0.12**247952 416.8
100 x 5	**0.01**241807 64.4	**0.01**241807 64.4	**0.01**288745 66.8	**0.01**288745 66.8
100 x 10	**0.01**213640 68	**0.01**213640 68	**0.01**278860 71.7	**0.01**278860 71.7
100 x 20	**0.06**258981 377.9	**0.06**258981 377.9	**0.06**618542 399.9	**0.06**618542 399.9
200 x 10	**0.01**021800 108.7	**0.01**047339 111.4	**0.01**068583 113.6	**0.01**042171 110.8
200 x 20	**0.03**283923 363.6	**0.03**572674 395.6	**0.04**053036 448.9	**0.03**684218 408
500 x 20	**0.00**762264 200	**0.01**173941 307.9	**0.01**435692 376.7	**0.00**912093 239.2
Średnio	**0.05**476425 192.758333	**0.05**536922 204.641666	**0.05**663365 218	**0.05**586796 202.9

[1)] Odchylenie względne O_{dolne}

[2)] Odchylenie bezwzględne B_{dolne}

Z tabeli wynika, ze najlepszy wynik otrzymano stosując pierwszą wersję algorytmu.

Tabela 4: Tabela średnich czasów rozwiązywania pojedynczego przykładu:
(na procesorze CeleronII 800 MHz, w milisekundach)

Grupa	NEH	mNEH	Prog. 1	Prog. 2	Prog. 3	Prog. 4
20 x 5	3	0	23	34	16	21
			23	34	19	24
20 x 10	0	0	77	88	87	82
			77	88	87	82
20 x 20	3	0	122	119	163	144
			122	119	166	147
50 x 5	0	60	138	137	198	181
			198	197	198	181
50 x 10	0	26	444	444	494	456
			470	470	494	456
50 x 20	5	22	818	911	1 073	973
			840	933	1 078	978
100 x 5	0	930	286	275	401	364
			1 216	1 205	401	364
100 x 10	8	2 030	1 241	1 260	1 680	1 494
			3 271	3 290	1 688	1 502
100 x 20	5	3 047	3 223	3 687	4 252	3 768
			6 270	6 734	4 257	3 773
200 x 10	30	9 856	6 903	4 371	4 695	8 145
			16 759	14 227	4 725	8 175
200 x 20	47	21 394	28 746	14 940	16 581	29 730
			50 140	36 334	16 628	29 777
500 x 20	401	195 762	916 846	123 072	103 259	694 143
			1 112 608	318 834	103 660	694 544

Pomiary czasów były wykonywane podczas działania programów (pomiar czasu był zawarty w kodzie programu). Niewielkie odchylenia wynikają z działania procesu zarządzającego przydziałem czasu procesora i konkurencją procesów o dostęp do zasobów systemu.
W pierwszym wierszu każdej komórki kolumny **Prog.** jest zamieszczony czas działania samego algorytmu (bez czasu wyznaczania rozwiązania początkowego), poniżej znajduje się sumaryczny czas działania algorytmu.

Pomimo zredukowania liczby iteracji podczas wyznaczania rozwiązania początkowego, w przypadku programu **Prog.2** czas wyznaczenia tego rozwiązania nadal dominuje nad czasem działania samego algorytmu.
Bezsprzecznie najlepsze wyniki dawał program **1**, jednak był on też najdłużej działającym programem. Kolejnymi najdłużej działającymi programami były programy **2** i **4**. Jak się okazuje lepsze wyniki (mierzone odchyleniem procentowym od dolnych ograniczeń) dawał program **Prog.2**. Dodatkowo
w przypadku większej niż 200 liczby zadań działał on także szybciej niż program **4** (ale dawał gosze wyniki dla dużej liczby zadań).

Tak więc jeżeli zależy nam na czasie obliczeń, wówczas należy dokonać wyboru pomiędzy pomiędzy wersjami **2** i **4** algorytmu. Najlepsze rozwiązania wyznacza wersja **1**. Dodatkowo, można jej zwiększyć liczbę iteracji podczas wyznaczanie rozwiązania początkowego jak też, ilości iteracji algorytmu tabu. Program numer **3** jest szybki, ale dość niedokładny. Może służyć jako generator rozwiązań początkowych dla innych wersji alorytmów popraw. Jednak użycie go w algorytmie lokalnych popraw typu ,,najlepszy z bloku" jest równoważne zwięszkeniu liczby iteracji w algorytmie tabu oraz zastosowania listy tabu o zmiennej długości.

9.3 Zmienna długość listy tabu

Jedną z możliwych modyfikacji algorytmów opisanych powyżej jest wprowadzenie listy tabu o zmiennej długości. Jak łatwo zauważyć, algorytmy dla pewnych długości listy tabu dają lepsze rezultaty niż dla innych. Jednak nie jest to regułą. Czy więc da się połączyć ,,zalety" róznych długości list tabu w jednym algorytmie?

Do testów wybrałem trójki określające listę tabu: *bazowa lista tabu*(**m**), *skrócona lista tabu*(**e**),*procentowy czas używania skróconej listy tabu*(**p**). Rozważono następujace wersje: 1m7e25p, 1m7e50p, 1m11e25p, 1m11e50p,
1m12e25p, 1m12e50p, 2m7e25p, 2m7e50p, 2m11e25p, 2m11e50p, 2m12e25p, 2m12e50p, **m** jest liczbą maszyn. Widoczne pogorszenie się wyników zarówno dla dłuższych niż 15 elementów list tabu o stałej długości jaki i dla dłuższych list tabu niż rozmiar zadania sugerowały niecelowość testowania innych możliwych kombinacji parametrów list tabu o zmiennej długości.

Rozwiązanie początkowe wyznaczono algorytmem **mNEH** o 500 iteracjach. Algorytm tabu wykonywał $max(500, n * log_{10}(n))$ iteracji.
Jeżeli początkowa długość listy tabu była mniejsza od długości skróconej listy tabu (na przykład dla zestawu testowego o 5 maszynach), wówczas lista ta nie ulegała zmianie.

Tabela 5: Wyniki testów dla listy tabu o zmiennej długości.

Identyfikator programu	$\mathbf{O_{dolne}}$	$\mathbf{B_{dolne}}$	$\mathbf{O_{górne}}$	$\mathbf{B_{górne}}$
1m7e25p	193.4083	**0.05**50925671	32.64166	**0.00**74949542
1m7e50p	**191.9833**	**0.0548526946**	**31.2166**	**0.0072604640**
1m11e25p	194.266	**0.05**53352802	33.5	**0.00**76965684
1m11e50p	**193.29166**	**0.0548448697**	**32.525**	**0.0072860458**
1m12e25p	193.333	**0.05**50409595	32.566	**0.00**74335575
1m12e50p	194.35833	**0.05**52191010	33.59166	**0.00**76067453
2m7e25p	193.8833	**0.05**54337787	33.1166	**0.00**77402795
2m7e50p	194.85833	**0.05**55090154	34.09166	**0.00**78438109
2m11e25p	195.35833	**0.05**57405558	34.59166	**0.00**80872042
2m11e50p	193.525	**0.05**58094541	32.75833	**0.00**81271116
2m12e25p	196.80833	**0.05**60655973	36.04166	**0.00**83507703
2m12e50p	194.94166	**0.05**59999353	34.175	**0.00**82964048
Średnio	194,16796166	**0,05**54119840	33,40131	**0,00**77686597

Najlepsze wyniki otrzymano algorytmem z początkową listą tabu równą ilości maszyn oraz o skracanej do 11 elementów (czyli o liście tabu równej dla zestawów odpowiednio 5, 10 i 20/11): 1m11e50p oraz algorytmem 1m7e50p, którego rozwiązania miały mniejszą średnią odległość względną od ograniczenia (a procentowo wyrażana średnia odległość rozwiązań wyznaczanych przez te programy różni się o 1 tysiączną procenta). W obu przypadkach skrócona lista tabu była używana w 50%.
Podobnie, jak w poprzednim przypadku lepsze okazują się być krótkie listy tabu. Zarówno średnie odległości bezwzględne są lepsze dla krótkich podstawowych list tabu długości 1m, jak i dla często używanych skróconych list tabu. Najgorsze wyniki otrzymano algorytmem mającym podstawową listę tabu o długości 2m skracaną do 12 elementów i używający w 3/4 przypadków swej podstawowej, długiej listy tabu.

9.4 Dłuższe przesunięcia elementów

Do tej pory rozważane były przesunięcia zadania bezposrednio ,,przed" i ,,za" blok, w którym znajdowało się zadanie.
W tym rozdziale sprawdzimy zachowanie się algorytmu, w którym przesunięcie następuje nie na bliższy koniec sąsiedniego bloku (czyli na koniec bloku poprzedzającego lub na początek bloku następnego), ale na dalszy koniec (początek bloku poprzedzającego i koniec bloku następnego).
Algorytm ten testowano dla listy tabu długości 5, 7, 11, 15, 20 i 25.

Tabela 6: Wyniki dla algorytmu ,,dłuższych przesunięć" dla różnych długości listy tabu.

Identyfikator programu	$\mathbf{O_{dolne}}$	$\mathbf{B_{dolne}}$	$\mathbf{O_{górne}}$	$\mathbf{B_{górne}}$
5dp	220.3	**0.06**07006887	59.5333	**0.01**28347884
7dp	209.95	**0.05**91032929	49.1833	**0.01**13288563
11dp	205.775	**0.05**83513163	45.00833	**0.01**06481707
12dp	205	**0.05**82689661	44.2333	**0.01**05572608
15dp	**204.45833**	**0.0581344369**	**43.69166**	**0.0104251619**
20dp	207.4166	**0.05**89082644	46.65	**0.01**11690568
25dp	207.4166	**0.05**91177135	46.65	**0.01**13279620
Średnio	209,090255	**0,05**90388937	48,32359	**0,01**12738477

Najlepsze wyniki otrzymano dla listy tabu o długości 15. Jednak były one gorsze od wyników poprzednio testowanych wersji algorytmu.

9.5 Jeszcze dłuższe przesunięcia

Przesunięcia na dalsze końce sasiedniego bloku nie dały poprawy wyników algorytmu. Sprawdzono także, czy jeszcze dłuższe przestawienia zadań w permutacji zależne od sąsiednich bloków moga dać poprawę wyników?
Dla otrzymania choć częściowej odpowiedzi na to pytanie przetestowano zachowanie sie algorytmu, w którym przestawienia następują na bliższy koniec drugiego sąsiedniego bloku (na koniec bloku poprzedzającego sąsiada z przodu i na początek kolejnego bloku po następnym) z listami tabu długości 7, 11, 15 i 20.

Tabela 7: Wyniki testów dla algorytmu ,,jeszcze dłuzszych przesunięć" dla różnych długości listy tabu.

Identyfikator programu	O_{dolne}	B_{dolne}	$O_{górne}$	$B_{górne}$
7jdp	215.00833	**0.06**09199140	54.24166	**0.01**31165072
11jdp	215.075	**0.06**09212416	54.30833	**0.01**31109323
15jdp	213.8166	**0.06**07526884	53.05	**0.01**29462904
20jdp	215.5166	**0.06**12532155	54.75	**0.01**33941308
Średnio	214.8541325	**0.06**09617648	54.0874975	**0.01**31419651

W wyniku testów okazało się, że jeszcze dłuższe przesunięcia elementów nie dają poprawy wyników.

9.6 Zmiana list tabu, dla długich przesunięć

Dość nieoczekiwane polepszone wyniki algorytmów dla dłuższych list tabu (o długości 15, a nie jak poprzednio 7 - 11) z punktu 9.4 *,,Dłuższe przesunięcia"* spowodowały, że sprawdzono, czy zmienna długość list tabu zastosowana (bez powodzenia w porównaniu ze stałą długością, ale z powodzeniem przy wyznaczanej zależnie od ilości maszyn długości list tabu) w punkcie 9.3 może poprawić działanie algorytmu.
Zastosowano listy tabu (posługując się notacją z rozdziału 9.3) 1m11e25p, 1m11e50p, 1m15e25p, 1m15e50p oraz 2m11e25p, 2m11e50p, 2m15e25p, 2m15e50p dla algorytmów opisanych w punktach 9.4 *,,Dłuższe przesunięcia elementów"*. Algorytmy z punktu 9.5 *,,Jeszcze dłuższe przesunięcia"* dawały tak słabe wyniki, że ewentualna poprawa jakości ich wyników, którą zresztą widać w tabeli przy porównaniu odpowiednich wersji algorytmu, o wartości około 0.01% nie jest warta czasu obliczeniowego poświęconego dla testowania tych algorytmów.

Tabela 8: Wyniki testów algorytmu ze zmienną dłuogścią listy tabu i dłuższymi przesunięciami.

Identyfikator programu	O_{dolne}	B_{dolne}	$O_{górne}$	$B_{górne}$
1m11e25pdp	207.64166	**0.05**84462393	46.875	**0.01**07408234
1m11e50pdp	205.525	**0.05**82971550	44.75833	**0.01**05867740
1m12e25pdp	206.55833	**0.05**83433747	45.79166	**0.01**06373013
1m12e50pdp	**204.90833**	**0.0580184371**	**44.14166**	**0.0103409471**
2m11e25pdp	206.866	**0.05**85880806	46.1	**0.01**08641239
2m11e50pdp	205.066	**0.05**83756636	44.3	**0.01**06836757
2m12e25pdp	206	**0.05**84935186	45.233	**0.01**07660468
2m12e50pdp	205.625	**0.05**86938353	44.85833	**0.01**09318806
Średnio	206,02395833	**0,05**84070380	45,25729	**0,01**06939466

Podobnie jak w podrozdziale 9.3, także w tym przypadku zastosowanie list tabu o zmiennej długości nie spowodowało znacznego polepszenia wyników algorytmów. Jednak w rozdziale 9.3 ta poprawa miała znaczenie, gdyż dotyczyła algorytmów które dawały dobre wyniki.

9.7 Wprowadzenie dodatkowych przesunięć

Innym sposobem polepszenia wyników algorytmów opisanych powyżej (w rodziałach 9.4 i 9.5) może być wprowadzenie dodatkowych przesunięć zadań.
Jeżeli podczas pewnej liczby iteracji nie nastąpiła poprawa (czyli nie znaleziono nowego, lepszego rozwiązania) obok standardowych, dla metody przesunięć elementów (czyli na dalszy koniec bloku sąsiedniego lub za blok sąsiedni) wprowadzę dodatkowo także kilkukrotne przesunięcia takie jak w rozdziale 9.1 (wykonane zostaną przesunięcia elementów na bliższy koniec sąsiedniego bloku).
Metodę tą przetestowano, dla dających najlepsze wyniki wersji algorytmu z paragrafów 9.4 *,,Dłuższe przesunięcia"* (dla algorytmu z listą tabu długości 11 i 15 elementów oznaczonych 11dp i 15dp, oznaczonych w tabeli wyników jako odpowiednio 11dpXp i 15dpXp, gdzie X oznacza ilośc dodatkowych przesunięć), oraz dla algorytmu z paragrafu 9.6 *,,Zmiana list tabu, dla długich przesunięć"* ze zmienną listą tabu o parametrach 1m12e50p. W tabeli wyników

wersja ta zostanie oznaczona jako dpzt**X**p. Każda z wersji algorytmu była testowana z dodatkowymi dwoma, trzema i pięcioma przesunięciami, po każdym przesunięciu ewentualny najlepszy wynik był zapamiętywany. Dodatkowe przesunięcia miały miejsce po 49 krokach iteracji bez poprawy najlepszego rozwiązania.

Tabela 9: Wyniki testów programów z dodatkowymi przesunięciami.

Identyfikator programu	$\mathbf{O_{dolne}}$	$\mathbf{B_{dolne}}$	$\mathbf{O_{górne}}$	$\mathbf{B_{górne}}$
11dp2p	198.225	**0.05**64545471	37.45833	**0.00**88145693
11dp3p	198.525	**0.05**64734498	37.75833	**0.00**88230439
11dp5p	**196.9**	**0.0558539854**	**36.133**	**0.0082476038**
15dp2p	198.825	**0.05**62908934	38.05833	**0.00**86715687
15dp3p	199.366	**0.05**65211473	38.6	**0.00**88662551
15dp5p	197.30833	**0.05**63178848	36.54166	**0.00**86873829
dpztdp2	198.05833	**0.05**62579646	37.29166	**0.00**86389666
dpztdp3	198.63333	**0.05**64521882	37.866	**0.00**88104409
dpztdp5	196.525	**0.05**59428967	35.75833	**0.00**83374002
Średnio	198,0406655	**0,05**62849952	37,27396	**0,00**86552479

Najlepsze rezultaty osiągnął algorytm **11dp5p** - 11-to elementowa lista tabu, przesunięcia na dalszy koniec bloku sąsiedniego oraz 5 dodatkowych przesunięć jeżeli w zadanej ilości iteracji nie nastąpiła poprawa wyniku.

9.8 Porównanie metod

Z każdej z 6 metod opisywanych w podrozdziałach rozdziału 8 wybrałem, zależnie od osiąganych rezultatów, jedną lub dwie najlepsze opisywane wersje algorytmu. Dodatkowo, z podrozdziału 9.2.1 wybrałem dla porównania wszystkie opisane tam wersje algorytmu.

Szczegółowe wyniki testów trzech najlepszych programów, czyli programów **7e** i **11e** (programy o stałej liście tabu) oraz programu o zmiennej liście tabu **1m7e50p** z rozdziału 9.3 zostały zamieszczone w tabeli 14 w rozdziale 11 *„Wyniki obliczeniowe”*.

Tabela 10: Zestawienie wyników najlepszych wersji algorytmu.

Identyfikator programu	O_{dolne}	B_{dolne}	$O_{górne}$	$B_{górne}$
1m (9.1)	194.35833	**0.05**54156989	33.59166	**0.00**77708465
7e (9.1)	193.44166	**0.05**47228463	32.675	**0.00**71627214
11e (9.1,9.2.1)	192.75833	**0.05**47642524	31.99166	**0.00**71906256
v2 (9.2.1)	204.64166	**0.05**53692246	43.875	**0.00**77867652
v3 (9.2.1)	218	**0.05**66336570	57.233	**0.00**90177374
v4 (9.2.1)	202.9	**0.05**58679670	42.133	**0.00**82625081
1m7e50p (9.3)	191.9833	**0.05**48526946	31.2166	**0.00**72604640
1m11e50p (9.3)	193.29166	**0.05**48448697	32.525	**0.00**72860458
15dp (9.4)	204.45833	**0.05**81344369	43.69166	**0.01**04251619
15jdp (9.5)	213.8166	**0.06**07526884	53.05	**0.01**29462904
1m12e50pdp (9.6)	204.90833	**0.05**80184371	44.14166	**0.01**03409471
11dp5p (9.7)	196.9	**0.05**58539854	36.133	**0.00**82476038

10 Opis programów

Na załączonej do pracy płycie CD zamieściłem sprawozdania z testów przeprowadzonych podczas pisania pracy (sprawozdania w plikach pdf oraz pliki wynikowe dla poszczególnych zestawów), oraz kody źródłowe programów w języku C++ (własności programów można ustalić ustawiając odpowiednie wartości zmiennych w nagłowkach programów. Zamieszczone są na płycie także źródłowe programów, w wersjach które dawały najlepsze wyniki podczas testów).
Programy były pisane i testowane na komputerze wyposażonym w procesor Celeron 800 MHz, oraz na komputerach w pracowniach Instytutu Informatyki.

Każdy program pobiera dane ze standardowego wejścia i wypisuje swoje wyniki na standardowe wyjście.
Zależnie od ustawienia zmiennych sterujących programem może wypisywać rozwiązanie początkowe wyliczone przez algorytm NEH (domyślnie ta opcja jest wyłączona).

Format danych wejściowych to:

```
ilosc_zadan ilosc_maszyn
czasy przetwarzania zadania 1 na maszynach od 1 do ilosc_maszyn
czasy przetwarzania zadania 1 na maszynach od 1 do ilosc_maszyn
...
czasy przetwarzania zadania ilosc_zadan na maszynach od 1 do ilosc_maszyn
```

Uruchomienie programu następuje z linii komend, dla programu z rozdziału 9.1 będzie to:

```
9-1_nzb.exe << plik_z_danymi > plik_wynikowy
```

11 Wyniki obliczeniowe

Do testów zastosowano zestawy testowych opisanych w pracy [3] na stronach 5-8.

Tabela 11: Odległości wyników wersji algorytmu NEH od dolnych ograniczeń:

Tabela 12: Porównanie wyników algorytmu mNEH z niemodyfikowanym algorytmem NEH:
według wzoru

$$-1 * \frac{C_{mNEH} - C_{NEH}}{C_{NEH}}$$

Wartość w kolumne oznacza średnią poprawę bezwzględną rozwiązań w danej guupie problemów.
W przypadku, gdy nie było zadań o takim samym całkowitym czasie wykonywania, algorytm działał z takim samym czasem. Natomiast czas jego działania zwiększał się jedynie w przypadku wystapienia w zestawie grup zadań o takim samym całkowitym czasie wykonywania, wtedy także można oczekiwać lepszego wyniku.
W kolumne **Lepszych** w tabeli została umieszczona ilość zestawów z danej grupy (grupy liczą 10 zadań), w których algorytm mNEH znalazł lepsze rozwiązanie od algorytmu NEH.

Tabela 13: Zestawienie wyników różnych wersji algorytmów NEH dla zestawów danych testowych, dla których nastąpiła zmiana wartości funkcji celu (dla pozostałych zestawów wyniki algorytmów są takie same):

W sumie poprawa nastapiła w przypadku 72 przykładów na 120 policzonych, największe poprawy daje algorytm mNEH z 1000 iteracji, choć mNEH z 500 iteracjami jest gorszy o 0.02%, a czas jego działania może być 2 razy krótszy.

Tabela 14: Zestawienie wyników najlepszych algorytmów oraz najlepszych znalezionych wyników:
(wyniki dla programów pochodzą z zestawów **7e_2**, **11e_1** oraz **1m7e50p_1**)

W kolumnie **Najlepsze** obok znalezionego najlepszego wyniku w nawiasach jest podany identyfikator testu, w którym wynik został znaleziony. Dokładną permutację mozna znaleźć na dołączonej płycie w pliku **raport**_identyfikator**.pdf** w katalogu **Raporty**, plik wynikowy zawierający odnalezioną permutację znajduje się w katalogu odpowiadającym rozdziałowi, w którym był testowany program o podanym identyfikatorze. Numery rozdziałów można także odczytać z poniższej tabelki:

Tabela 15: Identyfikatory rozdziałów i wersji programów.

Identyfikator	Rozdział
1m, 2m, 3m, 4m, 5m, 6m	9.1 *,,Najlepszy z bloków"*
5e, 6e, 7e, 8e, 9e, 10e, 11e, 12e, 13e, 14e, 15e, 20e, 25e, 50e, 6xe	9.1 *,,Najlepszy z bloków"*
v1, v2, v3, v4	9.2.1 *,,Wersja ostateczna algorytmu"*
1m7e25p, 1m7e50p, 1m11e25p, 1m11e50p, 1m12e25p, 1m12e50p, 2m7e25p, 2m7e50p, 2m11e25p, 2m11e50p, 2m12e25p, 2m12e50p,	9.3 *,,Zmienna długość listy tabu"*
5dp, 7dp, 11dp, 12dp 15dp, 20dp, 25dp	9.4 *,,Dłuższe przesunięcia elementów"*
7jdp, 11jdp, 15jdp, 20jdp	9.5 *,,Jeszcze dłuższe przesunięcia"*
1m11e25pdp, 1m11e50pdp, 1m12e25pdp, 1m12e50pdp, 2m11e25pdp, 2m11e50pdp, 2m12e25pdp, 2m12e50pdp	9.6 *,,Zmiana list tabu dla długich przesunięć"*
11dp2p, 11dp3p, 11dp5p 15dp2p, 15dp3p, 15dp5p dpztdp2, dpztdp3, dpztdp5	9.7 *,,Wprowadzenie dodatkowych przesunięć"*

12 Podsumowanie

W pracy zaprogramowano i przetestowano 63 wersje algorytmów korzystających z techniki lokalnych popraw działających według heurystyki tabu-search. Programy należały do sześciu grup, w których zostały zastosowane różne techniki popraw rozwiązania. W każdej z grup zostało przetestowane zachowanie się algorytmu zależne od długości listy tabu i techniki popraw zastosowanych w algorytmie.
Najlepsze programy osiągnęły błąd rzędu 0,7 % względem najlepszych znanych rozwiązań, średnia odległość od najlepszego znalezionego rozwiązania wynosiła poniżej 32 jednostek czasu dla wszystkich 120 zestawów.
Dodatkowo, znalazłem i wykorzystałem nieścisłość w teoretycznych założeniach algorytmu NEH, dzięki której zmodyfikowany algorytm NEH znalazł dwa

rozwiązania dla zestawów testowych lepsze od najlepszych znanych rozwiązań, oraz ogólnie średnia poprawa rozwiązań 120 zestawów testowych wynosiła 0,5%.
Podczas testowania programów zostało poprawionych (znalezionych lepszych niż załączone do zestawów danych testowych) 16 rozwiązań. Wartości tych rozwiązań te zostały przedstawione w tabeli 14 rozdziału 11 *,,Wyniki obliczeniowe"* w kolumnie C_{naj}.
Zważywszy na to, że najlepsze wyniki dla danych testowych były wyzaczone przez różne algorytmy, osiągnęty margines błędu rzędu 0,7% oraz znalezienie kilku lepszych rozwiązań niż podane wydaje się być spełnieniem założeń pracy, polegających na napisaniu programu znajdującego dobre rozwiązania problemu flow-shop.

Literatura

[1] J. Grabowski, J. Pempera. *New block properties for the permutation flow shop problem with application in tabu search.* **Joural of the Operational Research Society (2001) 52**, 210–220, 2001 Operational Research Society Ltd.

[2] E. Taillard. *Some efficient heuristic methods for the flow shop sequencing problem.* **Ecole polytechnique federale de Lausanne**, Octobre 1988, Revision mars 1989, ORWP 88/12.
http://www.eivd.ch/ina/collaborateurs/etd/articles.dir/flow_shop.ps

[3] E. Taillard. *Benchmarks For Basic Scheduling Problems.* ORWP89/21 Dec. 1989
http://www.eivd.ch/ina/collaborateurs/etd/articles.dir/benchmark.ps