Inteligencja Obliczeniowa

Projekt II: Szeregowanie zadań w permutacyjnym systemie przepływowym

AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Projekt realizowali:

Michał Bubel
Katarzyna Czyż
Ewa Deoniziak
Justyna Zbiegień

Spis treści

Wstęp

Optymalizacja jest poszukiwaniem najlepszego rozwiązania spośród wszystkich możliwych, jednak w możliwie jak najkrótszym czasie, dlatego rozwiązanie optymalne niekoniecznie musi być rozwiązaniem najlepszym. Aby móc zacząć poszukiwania tego rozwiązania często wykorzystuje się ogólne algorytmy zwane metaheurystykami, które służą do rozwiązywania zadań obliczeniowych optymalizacyjnych. Do jej algorytmów możemy zaliczyć algorytmy ewolucyjne, systemy rozmyte czy sztuczne sieci neuronowe, a także metodę Wspinaczki, Symulowanego Wyżarzania czy Tabu Search, algorytm zachłanny NEH oraz algorytm genetyczny z wykorzystaniem metody ruletki i metody turniejowej, które będą obiektami badań w projekcie.

Algorytmy ewolucyjne, czyli EA, są algorytmami opartymi na zasadach inspirowanych ewolucją. Ewolucja to nic innego jak proces ciągłego dostosowywania się żywych organizmów do panujących warunków, następuje więc przy tym eliminacja tych organizmów, bądź wyciszanie tych cech, które są nie przydatne, przy jednoczesnym promowaniu tych, które są korzystniejsze w danych warunkach. Algorytmy ewolucyjne nie gwarantują znalezienia optimum globalnego, ale zapewniają rozwiązanie, które będzie wystarczająco dobre w akceptowalnym czasie. Algorytmy ewolucyjne są wykorzystywane w algorytmach i programowaniu genetycznym oraz w strategiach i programowaniu ewolucyjnym. Jednak najbardziej rozwiniętą dziedziną EA są algorytmy genetyczne, do których zaliczamy metodę ruletki, jak i metodę turniejową, które wykorzystamy w naszym projekcie. Jedynym innym algorytmem, który został wykorzystany w projekcie jest algorytm NEH, który jest jednym z algorytmów zachłannych. Jego specyficzne działanie zostało wytłumaczone w osobnym rozdziale.

Wszystkie kody do algorytmów zostały napisane w języku *Visual Basic of Appliacation* w programie Excel z pakietu Microsoft. Algorytmy zostały sprawdzone na trzech zestawach danych, w których kolejno znajduje się 50, 100 i 200 zadań wykonywanych na kilku maszynach. Wynikiem każdego algorytmu jest czas w jakim zakończyć ostatnie zadanie ostatnia maszyna. Istotnym faktem jest, że kody VBA zostały puszczone na wcześniej odpowiednio utworzonej tabeli, która wykorzystuje takie funkcje jak: *Wyszukaj.Pionowo* w celu znalezienia numeru zadania z tabeli pierwotnej oraz funkcja *Maks*, do sprawdzenia, w której komórce znajduje się większa wartość (późniejszy czas zakończenia pracy) – zakończenie obecnie sprawdzanego zadania z poprzedniej maszyny czy zakończenie pracy obecnie sprawdzanej maszyny ale na poprzednim zadaniu. Tabele z ułożonymi zadaniami po przejściu algorytmów, znajdują się w osobnych trzech plikach Excel – po jednym pliku na każdy zestaw danych (50, 100, 200 zadań).

1. Metoda wspinaczki (z multistartem)

Opis metody

Metoda wspinaczki, czyli Hill Climbing, jest jedną z podstawowych metod metaheurystyki. Metodę można również spotkać pod nazwą algorytmu największego wzrostu. Metoda ta nie buduje rozwiązania kawałek po kawałku, lecz daje wynik w całości – każdy punkt, który zostaje poddany

analizie jest pełnym rozwiązaniem problemu, może być on lepszym bądź gorszym, które następnie jest lokalnie poprawiane, czyli w najbliższym sąsiedztwie. Jako zaletę metody wspinaczki można wymienić prostą jej implementacje, szybkie działanie oraz małe wymagania pamięciowe. Wadą jej jest duża zależność końcowego wyniku od początkowego punktu startu oraz uzyskiwanie takich rozwiązań, których nie da się poprawić dzięki prostej modyfikacji, ponieważ są one wynikiem (minimum) lokalnym. Algorytm natyka problemy również z określeniem kierunku poszukiwań, gdy na dużym obszarze wartość funkcji jest stała lub zmienia się nieznacznie.

Algorytm wspinaczki polega na rozpoczęciu poszukiwań z losowo wybranego punktu, a następnie przeanalizowaniu jego sąsiadów i wybraniu tego, który ma największą wartość. Wartość ta jest wyborem najlepszego rozwiązania jako początkowego do następnej iteracji, którą należy powtórzyć kilka razy (zalecane jest 2-4), aż znalezienie lepszego rozwiązania nie będzie możliwe.

Metoda Wspinaczki w kodzie

W metodzie Wspinaczki, jak wspomniano wyżej w opisie metody, najpierw wybiera się losową liczbę (zadanie) i sprawdza się jej sąsiedztwo. W projekcie mamy do czynienia z trzema różnymi zestawami danych, gdzie jest różna liczba zadań, które wykonywane są na kilku maszynach. W zadaniach mamy do czynienia z 50, 100 i maksymalnie 200 zadaniami, dlatego założono, że sąsiedztwem losowego zadania jest każde inne zadanie wybrane z pozostałych. Dzięki temu założeniu kod może początkowo losować, nie jedno, lecz dwa zadania i sprawdzić ich zamianę. Program zapisuje wcześniejszy wynik i zamienia wylosowane zadania. Jeśli nowszy wynik okaże się być lepszy, program zostawia daną zamianę i od nowa losuje dwa zadania, jednak jeżeli wynik jest gorszy od wcześniejszego, dane zadania zostają zamienione na swoje pierwotne pozycje a program ponawia cały proces. Aby program nie utknął w optimum lokalnym wykorzystuje się tzw. multistart. Całość powtarza się w pięciu pętlach, przez co algorytm za każdym razem startuje z innego miejsca i dopiero program kończy swoje działanie. W programie zastosowano kolejno: 1.000, 5.000 oraz 10.000 iteracji (zamian) w każdej z pięciu pętli. Działanie to ma na celu sprawdzenie czy wynik metody Wspinaczki zależy od liczby jakie wykona. Algorytm do każdej liczby iteracji powtórzono wielokrotnie (co najmniej 10 razy), aby znaleźć najlepszy wynik i aby dokonać analizy hipotez. Kolejnym parametrem, który sprawdzono jest multistart. Sprawdzenie tego parametru polega na puszczeniu kod w pętli 2,5 oraz 6 razy, ponieważ z każdym początkiem pętli zadania są ułożone według „poprzedniej” pętli, a co za tym idzie algorytm zaczyna działanie z różnych miejsc.

Wynik kodu i analiza

Hipoteza: wynik algorytmu w dużej mierze zależy od liczby iteracji (im większa liczba iteracji tym dokładniejszy wynik); wynik algorytmu w dużej mierze zależy od liczby powtórzeń (multistartu).

Program opierający się na powyższych założeniach otrzymał następujące wyniki:

Wyniki ze względu na zmianę liczby iteracji (dla 5 powtórzeń):

50 zadań:

Dla 1000 iteracji	Dla 5000 iteracji	Dla 10000 iteracji
Wynik: 3202	Wynik: 3202	Wynik: 3202

100 zadań:

Dla 1000 iteracji	Dla 5000 iteracji	Dla 10000 iteracji
Wynik: 6605	Wynik: 6507	Wynik: 6497

200 zadań:

Dla 1000 iteracji	Dla 5000 iteracji	Dla 10000 iteracji
Wynik: 12096	Wynik: 11879	Wynik: 11795

Z powyższych tabel zawierających wyniki algorytmu wynika, że liczba iteracji nie ma takiego istotnego wpływu na wynik. Należy też zwrócić uwagę na zależność, że im więcej zadań, tym ten wpływ jest większy. Dla zestawu danych, gdzie jest tylko 50 zadań, wynik jest identyczny dla każdej liczby iteracji, jednak przy zestawie z 200 zadaniami różnica między 1.000 a 10.000 iteracji wynosi niemal tysiąc.

Wyniki ze względu na liczbę powtórzeń - multistart (dla 1000 iteracji):

50 zadań:

Dla 2 powtórzeń	Dla 5 powtórzeń	Dla 6 powtórzeń
Wynik: 3210	Wynik: 3202	Wynik: 3202

100 zadań:

Dla 2 powtórzeń	Dla 5 powtórzeń	Dla 6 powtórzeń
Wynik: 6781	Wynik: 6705	Wynik: 6676

200 zadań:

Dla 2 powtórzeń	Dla 5 powtórzeń	Dla 6 powtórzeń
Wynik: 12309	**Wynik: 12016**	**Wynik: 11933**

Z powyższych tabel, przedstawiających wyniki w zależności od multistartu wynika, że multistart ma mały wpływ na wynik. W przeciwieństwie, do zmiany parametru iteracji, dla zestawu z 50 zadaniami wynik nie jest identyczny – jest mała różnica między 2 a 5 powtórzeniami. Jednak na podobieństwo wyniku dla zmiennego parametru iteracji, im większa liczba zadań tym większa jest różnica w wyniku dla różnej liczby powtórzeń.

Biorąc pod uwagę wyżej postawione hipotezy oraz otrzymane wyniki, można powiedzieć, że obie się nie potwierdziły. Jak się okazało liczba iteracji nie ma tak dużego wpływu na wynik, jak to było w zestawach danych z tylko jedno maszyną. Hipoteza mówiąca o dużym wpływie multistartu na wynik, również w dużej mierze się nie potwierdziła – ma on jeszcze mniejszy wpływ na wynik niż liczba iteracji.

2. Metoda symulowanego wyżarzania

Opis metody

Symulowane wyżarzanie (SA) , jak sama nazwa wskazuje, ma wiele wspólnego ze zjawiskiem wyżarzania w metalurgii. Wyżarzanie jest to jedna z operacji obróbki cieplnej metali, która polega na nagrzaniu materiału do określonej temperatury, podtrzymaniu jej, a następnie powolnym jej zmniejszaniu. Celem wyżarzania jest przybliżenie stanu materiału do warunków równowagi, by np. uzyskać specyficzne jego cechy, takie jak twardość. Metal posiada na początku niewielkie mankamenty, które osłabiają jego całą strukturę. Dzięki ogrzewaniu, a następnie studzeniu pozbywamy się ich. Temperatura początkowa ogrzewania nie może być zbyt niska, a studzenie musi być kontrolowane i powolne, aby nie osiągnąć odwrotności zamierzonego efektu. Symulowane wyżarzanie jest jedną z metod przeszukiwania sąsiedztwa, w której zakładamy, że dla każdego elementu rozwiązań potrafimy wyznaczyć wartość funkcji oceniającej jakość rozwiązania. Celem poszukiwań jest znalezienie minimum funkcji. Każdy punkt z przeszukiwanego obszaru jest postrzegany jako stan pewnego fizycznego obiektu, a minimalizowana funkcja jest analogiczna do stanu energii wewnątrz tego obiektu.

Algorytm symulowanego wyżarzania działa iteracyjnie zbliżając się do rozwiązania optymalnego, podobnie jak algorytm wspinaczki. Jeżeli nowe rozwiązanie jest lepsze to zostaje ono zatwierdzone jako nowe rozwiązanie, tak jak w algorytmie wspinaczki. Różnica się pojawia w

momencie gdy nowa propozycja jest gorsza od wcześniejszego - wybieramy rozwiązanie losując je. Zostaje wprowadzany parametr „temperatura”, który reguluje proces wyboru kolejnych rozwiązań i na początku jest on dużą wartością. Jeśli wartość parametru jest duża – istnieje duże prawdopodobieństwo wyboru nowego, lepszego rozwiązania, zaś jeśli parametr jest mały to prawdopodobieństwo wyboru nowego, ale gorszego rozwiązania, jest bardzo małe. W miarę poszukiwań wartość temperatury jest stale obniżana wraz z kolejnymi iteracjami, aż do otrzymania zerowej wartości, dzięki czemu w kolejnych przeszukiwaniach szansa przejścia do nowego, ale gorszego rozwiązania maleje.

Metoda Symulowanego Wyżarzania w kodzie

Metoda ta zaczyna się analogicznie do metody wspinaczki – losowane są dwa zadania spośród wszystkich. Program zapisuje wcześniejszy wynik i zamienia wylosowane zadania. Jeśli nowy wynik jest lepszy program zostawia zamianę i rusza do kolejnej pętli, gdzie powtarza cały proces. Natomiast jeżeli nowy wynik okazał się być gorszy (w tym kontekście większy) program przechodzi do funkcji *if()*, gdzie wyznacza „temperaturę”. „Temperatura” jest oznaczona wzorem:

$$e * \frac{Nowe\ rozwiązanie\ -\ Stare\ rozwiązanie}{100 * i}$$

gdzie *e* jest liczbą Eulera równą w przybliżeniu 2,71, a *i* oznacza numer aktualnie wykonywanej iteracji. Program losuje liczbę losową z przedziału 0 do 1 i porównuje ją z „temperaturą”. Jeśli „temperatura” okazała się być większa od losowej liczby program zamienia z powrotem zadania na ich miejsca pierwotne. Jak widać według tego programu im wyższa liczba iteracji, tym niższa jest „temperatura”, a co za tym idzie wysokie prawdopodobieństwo na znalezienie takiej kombinacji zadań, w której na pozór „gorsza” zamiana może się okazać zamianą słuszną mającą udział w ogólnie niższym wyniku. W programie zastosowano kolejno: 1.000, 5.000 oraz 10.000 iteracji (zamian) w każdej z pięciu pętli. Działanie to ma na celu sprawdzenie czy wynik metody Wyżarzania zależy od liczby jakie wykona. Algorytm do każdej liczby iteracji powtórzono wielokrotnie (co najmniej 10 razy), aby znaleźć najlepszy wynik i aby dokonać analizy hipotez. Kolejnym sprawdzonym parametrem jest funkcja temperatury. Zastosowano kolejno funkcje:

a. f(i) = i-1 , gdzie i ∈ (0,1000)
b. f(i) = i-3 , gdzie i ∈ (0,1000)
c. f(i) = i^(0.99) , gdzie i ∈ (1.1 , 10000)

Wynik kodu i analiza

Hipotezy: liczba iteracji nie ma dużego wpływu na wynik; funkcja temperatury (oraz jej dziedzina) ma ogromny wpływ na wynik.

Program opierający się na powyższych założeniach otrzymał następujące wyniki:

Wyniki ze względu na liczbę iteracji:

50 zdań:

Dla 1000 iteracji	Dla 5000 iteracji	Dla 10000 iteracji
Wynik: 3467	Wynik: 3433	Wynik: 3481

100 zadań:

Dla 1000 iteracji	Dla 5000 iteracji	Dla 10000 iteracji
Wynik: 7734	Wynik: 7575	Wynik: 7807

200 zadań:

Dla 1000 iteracji	Dla 5000 iteracji	Dla 10000 iteracji
Wynik: 12096	Wynik: 11879	Wynik: 11795

Z powyższych tabel można zobaczyć, że liczba iteracji ma wpływ na wynik, jednak nieznaczny. Największe różnice można spostrzec dla danych z 200 zadaniami, tam największa różnica w wynikach wynosi niecały tysiąc. Natomiast dla zestawu z 50 zadaniami różnica jest marginalna. Jednak należy zwrócić uwagę, że w zestawach z 50 oraz 100 zadaniami dla 10 tyś. iteracji wyniki są gorsze niż dla mniejszej liczby iteracji. Może to być spowodowane dużą losowością funkcji.

Wyniki ze względu na funkcję temperatury:

50 zdań:

I funkcja	II funkcja	III funkcja
Wynik: 3467	Wynik: 3559	Wynik: 3212

100 zadań:

I funkcja	II funkcja	III funkcja
Wynik: 7734	Wynik: 7738	Wynik: 7269

200 zadań:

I funkcja	II funkcja	III funkcja
Wynik: 12309	**Wynik: 12016**	**Wynik: 11933**

Z powyższych tabel wynika, że funkcja temperatury ma wpływ na wynik. Największe różnice widać dla funkcji potęgowej, czyli funkcji nr 3. Jednak należy zauważyć, że wyniki nie różnią się znacząco, największe różnice są, tak jak przy liczbie iteracji, dla zestawu z największą liczbą zadań, a najmniejsze dla zestawu z najmniejszą liczbą zadań.

Biorąc pod uwagę wyżej stwierdzone hipotezy oraz otrzymane wyniki można rzec, że tylko jedna z nich się potwierdziła. Liczba iteracji ma wpływ na wynik, jednak niewielki, tak jak to stwierdzono w hipotezie. Natomiast druga hipoteza zakładająca duży wpływ na wynik funkcji temperatury (oraz jej dziedziny) nie spełniła się. Funkcja i jej dziedzina mają wpływ na wartość wyniku, jednak nie większy niż liczba iteracji.

3. Metoda przeszukiwania Tabu Search

Opis metody

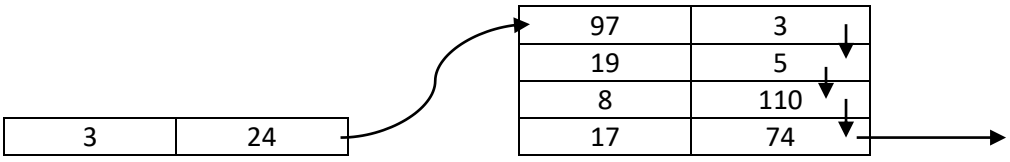
Tabu Search jest metaheurystyką, w której stosuje się nakładanie ograniczeń i wykorzystuje się je do znalezienia najlepszego rozwiązania. Tabu Search można określić jako metodę dynamicznej zmiany sąsiedztwa danego rozwiązania, które może się zmieniać w zależności od uzyskanych informacji na temat rozwiązywanego problemu. TS przypomina algorytm wspinaczki, a nawet niektórzy twierdzą, że TS jest jednym z wariantów wspinaczki, jednak podstawowa różnica między nimi polega na tym, że w TS mamy możliwość „odwiedzenia" sąsiadów, którzy posiadają gorsze wartości od aktualnie rozpatrywanych, więc możemy wyjść z maksimum lokalnego i dalej poszukiwać rozwiązań. W algorytmie wspinaczki nie można by tego zastosować, gdyż wpadlibyśmy w pętle – wychodząc z optimum lokalnego, a następnie od razu do niego wchodząc, a w TS lista tabu nam to uniemożliwia.

Algorytm TS polega na przejrzeniu przestrzeni rozwiązań, a następnie przechodzeniu do tego sąsiada, który posiada najwyższą wartość funkcji, o ile nie znajduje się on na liście punktów zabronionych, czyli tabu. Na liście tej są przechowywane *k* ostatnio odwiedzane punkty, które nie mogą zostać wykorzystane przez określoną liczbę iteracji. Zakończenie poszukiwań następuje po przekroczeniu limitu czasu, bądź wykonaniu określonych liczby iteracji.

Metoda TS w kodzie

Program bazuje na przejściu przez dwie zagnieżdżone pętle *for()*, w których wybiera po kolei zadanie i sprawdza jego zamianę z każdym pozostałym zadaniem. Dokonuje się to w analogiczny

sposób jak w powyższych programach. Program wybiera dwa zadania, zamienia je miejscami i porównuje zapisany wcześniej stary wynik (sumę kwadratów różnic czasu wykonania zadania i jego terminu) z nowopowstałym wynikiem. Jeśli jest on lepszy zapisuje nowy wynik jako stary i zapamiętuje kombinację zdań, jednak zamienia jest z powrotem na pierwotne miejsca. Program bierze kolejne zadanie i znowu sprawdza w ten sposób zamianę. Gdy program sprawdzi wszystkie kombinacje z pierwszym zadaniem (czyli kolejno 49, 149 lub 199 kombinacji, ponieważ eliminuje się kombinację zamiany zadania z samym sobą), wybiera zapisaną „najlepszą" kombinację z najniższym wynikiem i dokonuje permanentnej zamiany. Zamiana ta jest dodana pierwszego wiersza tablicy Tabu, czyli tablicy zakazanych zamian. Następnie program zwiększa zadanie o 1 i znowu powtarza algorytm, jednak od teraz sprawdza najpierw czy najbardziej optymalna zamiana nie znajduje się w tablicy Tabu, jeśli tak to nie wykonuj daje zamiany tylko następną z kolei najlepszą zamianę. Tablica tabu posiada dwie kolumny (ponieważ zamienia się ze sobą dwa zadania – jedno zadanie jest umieszcza w jednej kolumnie) i tyle wierszy, na ile ma być blokowana dana zamiana. Przy każdej nowej zamianie zawartość wierszy (czyli zamiany) przesuwają się o jeden wiersz w dół, aż w końcu z niej „wypadną" i zostaną wykreślone z listy Tabu, np.:



Według powyższego przykładu w aktualnej iteracji dokonuje się zamiany zadania 3 z zdaniem 24. Ta zamiana wędruje do pierwszego wiersza tabeli, a pozostałe wiersze przesuwają się o jeden w dół, oprócz wiersza ostatniego, w którym znajduje się zamiana zadania 17 z zadaniem 24. Ta zamian zostaje usunięta z listy Tabu, a co za tym idzie, została ona umożliwiona przy następnym wyborze zamiany. W programie zastosowano kolejno: 2, 5 oraz 10 iteracji (zamian) w każdej z pięciu pętli. Działanie to ma na celu sprawdzenie czy wynik metody Tabu zależy od liczby jakie wykona. Kolejnym parametrem jaki sprawdzono jest długość tablicy tabu. Sprawdzono jej wpływ na wartość wyniku dla tablic o długości 5, 20 i 500. Ostatnim zmiennym parametrem jest sposób początkowego ułożenia zadań w czasie startowania algorytmu.

Wyniki kodu i analiza

Hipoteza: liczba iteracji ma znaczny wpływ na wartość wyniku; długość tablicy tabu nie ma wpływu na wynik; początkowe ułożenie zadań nie wpływa na wynik.

Wyniki ze względu na liczbę iteracji:

50 zdań:

2	5	10
Wynik: 3202	**Wynik: 3202**	**Wynik: 3202**

100 zadań:

2	5	10
Wynik: 6743	Wynik: 6631	Wynik: 6631

200 zadań:

2	5	10
Wynik: 11852	Wynik: 11852	Wynik: 11852

Jak widać z powyższych tabel wartość wyniku nie zależy od liczby iteracji. Jedynie przy 2 iteracjach wyniki nieznacznie się różnią od tych, gdzie iteracji jest więcej. Jednak można stwierdzić, że nie widać istotnego wpływ liczby iteracji na wartość wyniki, zwłaszcza przy większej liczbie zadań.

Wyniki ze względu na długość tablicy tabu:

50 zdań:

5	20	500
Wynik: 3210	Wynik: 3202	Wynik: 3202

100 zadań:

5	20	500
Wynik: 6793	Wynik: 6793	Wynik: 6793

200 zadań:

5	20	500
Wynik: 11960	**Wynik: 11960**	**Wynik: 11960**

Jak widać na powyższych tabelach wyniki są identyczne, poza wynikiem dla długości tablicy tabu równej 5 dla zestawu z 50 zadaniami. Może oznaczać to, że wpływ na wartość wyniku długość tablicy tabu ma tylko przy małej liczbie zadań oraz przy krótkiej długości tablicy, w każdym innym przypadku ten parametr nie ma znaczenia.

Wyniki ze względu na początkowe ułożenie zadań:

50 zdań:

Bez zmiany	Pierwsza zmiana	Druga zmiana
Wynik: 3210	**Wynik: 3202**	**Wynik: 3142**

100 zadań:

Bez zmiany	Pierwsza zmiana	Druga zmiana
Wynik: 6743	**Wynik: 6615**	**Wynik: 6766**

200 zadań:

Bez zmiany	Pierwsza zmiana	Druga zmiana
Wynik: 11852	**Wynik: 11868**	**Wynik: 11897**

Jak widać w tabelach powyżej zmiana ułożenia początkowych zadań ma wpływ na wynik, ale nie zawsze jest to pozytywny wpływ. Przy 50 zadaniach każda zmiana uzyskała lepszy wynik, przy 100 tylko jedna zmiana dała lepszy wynik, zaś druga nie różniła się znacząco od wyniku uzyskanego bez zmian. Zaś przy 200 zadaniach zmiany dały gorsze wyniki niż miało to miejsce bez ruszania

początkowych zadań. Można wysunąć wnioski, że zmiana kolejności zadań ma wpływ na wynik przy mniejszej ilości zadań.

Możliwym jest rozszerzenie badania zmiany kolejności zadań na ostateczny wynik, przykładowo badając zmianę kolejności np. 10, 25, 50 i 75% ogólnych zadań lub sprawdzić czy zależy to jednak od jeszcze innego ułożenia początkowych zadań. Jednak biorąc pod uwagę czas trwania obliczeń (od 10 minut dla 50 zadań do ponad 2,5 godziny dla 200) nie zdążono dokładnie zbadać tego zagadnienia.

Zgodnie z postawionymi hipotezami można stwierdzić, że wszystkie zostały niepotwierdzone. Liczba iteracji nie ma wpływu na wartość ostatecznego wyniku, długość tablicy tabu już ma wpływ na wynik, ale podobnie jak z początkowym ułożeniem zadań – oba mają znaczenie, ale tylko przy mniejszej ilości zadań. W Tabu Search dla 50 zadań najlepszą opcją okazała się kolejność zadań policzona ze względu na zamianę kolejności początkowych zadań, tak samo jak i dla 100 zadań, zaś dla 200 jest to wynik uzyskany ze względu na liczbę iteracji.

4. Metoda NEH

Opis metody

Metoda NEH (*Nawaz, Enscore, Ham*) jest jednym z algorytmów zachłannych. Algorytm zachłanny to taki, który w każdym etapie (kroku) dokonuje się najlepszego (czyli zachłannego) rozwiązania częściowego. Czyli algorytmy tego typu szukają tzw. optimum lokalnego, przez co nie są w stanie znaleźć, w ogólnym rozrachunku, najoptymalniejszego rozwiązania.

Działanie algorytmu NEH polega na dodawaniu kolejnych zadań na następne puste pole w tabeli. Następnie sprawdzany wynik algorytmu w zależności od miejsca, które zajmuje to zadanie. Końcowy wynik, im mniejszy – typ lepszy. Jeśli wynik jest lepszy dokonana zamiana pozycji zadania zostaje zapisywana w pamięci, a zadania wracają na swoje pierwotne pozycje. Jak widać działanie algorytmu, tak jak algorytm Wspinaczki w dużej mierze zależy od początkowego układu zadań.

Metoda NEH w kodzie

Algorytm NEH optymalizacji zadań na wielu maszynach polega na uszeregowaniu zadań tak, aby zminimalizować czas ich wykonania, przy uwzględnieniu pewnych ograniczeń. Każde z zadań, polegających na wytworzeniu danego wyrobu odbywa się w kilku etapach, na kilku maszynach, w jednakowej kolejności. Każdy kolejny etap następuje po zakończeniu poprzedniego. Dodatkowo, jedna maszyna może pracować jednocześnie tylko nad jednym wyrobem.

Aby zoptymalizować pracę na wielu maszynach, należy w pierwszej kolejności wyznaczyć czas wykonania poszczególnych zadań, czyli sumy czasów pracy wszystkich maszyn do danego zadania. Następnie, na podstawie tych wartości należy uszeregować zadania w kolejności malejącej. Kolejnym krokiem jest wyznaczanie czasów zakończenia wykonania poszczególnych etapów, przy uwzględnieniu wcześniej wymienionych ograniczeń. Czas zakończenia wykonywania ostatniego etapu w ostatnim zadaniu jest czasem zakończenia całego procesu.

Algorytm zależy jedynie od jednego parametru jakim jest początkowy układ zadań. Gdyby nie układać zadań według malejącej sumy czasów poszczególnych maszyn to końcowy wyniki byłby inny. Jednak gdyby zaburzyć to początkowe ułożenie zadań algorytm stracił by sens, dlatego puszczono kod bez zmiany jakikolwiek parametrów. Jedyną kwestią pozostawioną do analizy jest sprawdzenie czy wynik zmienia się z każdorazowym puszczeniem kodu. Idąc tokiem myślenia, że kod nie wybiera losowych zadań (tak jak Wspinaczka z multistartem), tylko bierze zadania po kolei, wynik kodu nie powinien się zmieniać wraz z jego ponownym puszczeniem.

Wynik kodu i analiza

Hipoteza: wynik kodu zależy od liczby zadań

Wyniki:

50 zadań	100 zadań	200 zadań
Wynik: 3591	**Wynik: 7938**	**Wynik: 13783**

Patrząc na powyższą tabelę o raz wyżej stwierdzoną hipotezę, można wysnuć konkluzję, że wynik algorytmu NEH w dużej mierze zależy od liczby zadań. Wyniki rosną wprost proporcjonalnie do liczby zadań, co jednoznacznie potwierdza hipotezę.

5. Algorytmy genetyczne

Opis metody

Algorytmy genetyczne odwzorują procesy ewolucyjne dzięki mechanizmom genetycznym. Stosując je do znalezienia optymalnego rozwiązania zaczynamy od zebrania zestawu potencjalnych rozwiązań, które będą naszą populacją początkową. Każde rozwiązanie opisujemy za pomocą genomu, czyli sformalizowanego zapisu, który pozwoli nam na odtworzeniu jego cech, które pełnią u nas role genów. Wraz z postępem poszukiwań genomy będą ulegać zmianie tworząc nowe warianty. Jedną z zalet algorytmów genetycznych jest umiejętność badania wielu potencjalnych dróg, które prowadzą do określonego celu, dzięki czemu sprawdzają się w rozwiązywaniu problemów, które mogą mieć dużo możliwych rozwiązań.

Algorytm genetyczny polega na przejściach pomiędzy kolejnymi pokoleniami. Naszym początkowym krokiem jest więc zebranie i zakodowanie populacji początkowej. W każdym pokoleniu osobniki z populacji zostają poddane ocenie poprzez funkcję dopasowania. Funkcja ta określa jak dobrze każdy osobnik poradził sobie z postawionym przed nim zadaniem. Kluczową role pełni odpowiednie sformułowanie funkcji dopasowania – to właśnie ona zadecyduje, w którym kierunku rozwinie się populacja początkowa. Do następnego pokolenia zostanie wybrana tylko ta część populacji, która została najlepiej oceniona. W następnym kroku zostają one poddane operatorom mutacji i skrzyżowania, dzięki którym zostaną przeprowadzone odpowiednie modyfikacje i wymieszanie puli genowej. Nowe pokolenie będzie składać się ze zmodyfikowanych wariantów. Tych kilka kroków

powtarzamy przez kilka kolejnych pokoleń, aż do przekroczenia progu wartości, który zostaje zwracany przez ustaloną funkcję dopasowania.

W tym projekcie skupimy się na metodzie ruletki oraz metodzie turniejowej.

Metoda turniejowa polega na ustaleniu liczby turnieju. Następnie dzielimy populację na dowolną ilość grup, w której również sami decydujemy o liczności. W każdej z takich grup wybieramy takiego osobnika, który jest najlepiej przystosowany z wszystkich należących do danej grupy. Zatem do populacji rodzicielskiej zostaną wybrane najlepsze osobniki. Ta metoda jest skuteczna zarówno w minimalizowaniu problemów, jaki i w ich maksymalizacji.

Metoda ruletki, jak sama nazwa wskazuje polega na losowaniu obiektu. Można to w prosty sposób zwizualizować - koło ruletki dzielimy na pewną liczbę wycinków, które nie muszą być identyczne, a każdy obszar przyporządkowujemy wielkością wprost proporcjonalnie do jakości osobnika. Następnie koło ruletki zostaje wprawione w ruch, a po jego zatrzymaniu to wskaźnik decyduje, który osobnik wchodzi do grupy rozrodczej. Oczywiście, im większy obszar został przypisany osobnikowi – tym istnieje większa szansa jego wylosowania. W przypadku naszego projektu prawdopodobieństwo wylosowania jest to suma jednego zdarzenia podzielona przez sumę wszystkich zdarzeń.

Warto tu wspomnieć również o takich pojęciach jak krzyżowanie i mutacja. Krzyżowanie to losowe dobieranie chromosomów w pary – powstaje on a w wyniku losowego przecięcia dwóch chromosomów w danym punkcie, a następnie zamianie podzielonych części pomiędzy nimi. Każda taka para tworzy nowe chromosomy, które nazywane są potomkami. Mutacja zaś polega na zamianie na przeciwny dany chromosom, a stosuje się ją w celu wprowadzenia różnorodności populacji, zazwyczaj występuje ona z bardzo małym prawdopodobieństwem: 0,01.

Metoda algorytmu genetycznego w kodzie

Na początku wymieniane jest między sobą 30 zadań i zapisywanie ich kolejności oraz sumy do tablicy osobników. W ten sposób tworzonych jest 250 osobników. Następnie wykorzystywana jest selekcja metodą turniejową lub ruletki. W pierwszej metodzie losowanych jest dwóch z 250 osobników, a następnie osobnik o lepszej funkcji dopasowania jest dopisywany do tablicy rodziców. W drugiej zaś każdy osobnik znajduje się w przypisanym przedziale dystrybuanty. Do każdego rodzica przypisywana jest liczba losowa, która przypisuje do rodzica osobnika w zależności w jakim przedziale dystrybuanty się znajduje. Następnie wykonywane jest krzyżowanie. W metodzie PMX losowana jest liczba, a następnie tworzona jest sekcja kojarzenia, gdzie dolny przedział to: 1 + wylosowana liczba 2, górny przedział natomiast to: liczba zadań – wylosowana liczba. W kolejnym kroku następuje uzupełnienie pozostałych genów. Jeżeli dany gen już wystąpił to zostaje przypisany do tablicy wyrazów wolnych, gdzie po uzupełnieniu genów niepowtarzających się trafi na najbliższe od końca wolne miejsce. W metodzie operatora losowego z uzupełnianiem losowana jest liczba, która odpowiada za wymianę genów pary rodziców. Następnie dla tej pary losowane są miejsca wymiany tychże genów. Do tak powstałych potomków w taki sam sposób zostaje przypisana tablica wyrazów wolnych, gdzie po uzupełnieniu genów niepowtarzających się wyraz wolny z początku tablicy trafi na najbliższe od końca wolne miejsce. Kolejnym krokiem algorytmu jest mutacja, gdzie do każdego

potomka losowana jest liczba od 0 do 1 i jeżeli będzie ona mniejsza od 0,05 to nastąpi wymiana dwóch zadań potomka między sobą. Kolejne pokolenie jest wyznaczane dzięki przypisaniu powstałej tablicy potomków do tablicy osobników i powtórzeniu algorytmu od metody ruletki lub turniejowej.

Wynik kodu i analiza

Hipoteza: wynik kodu zależy od liczby potomków

Metoda ruletki razem z krzyżowaniem PMX:
Dla 50 zadań :

3 pokolenia	5 pokoleń	10 pokoleń
Wynik: 3579	**Wynik: 3579**	**Wynik: 3536**

Dla 100 zadań:

3 pokolenia	5 pokoleń	10 pokoleń
Wynik: 7412	**Wynik: 7412**	**Wynik: 7412**

Dla 200 zadań:

3 pokolenia	5 pokoleń	10 pokoleń
Wynik: 13443	**Wynik: 13443**	**Wynik: 13443**

Wyniki odpowiadają za najlepszego dotychczas stworzonego potomka. Z powyższych tabel wynika, że liczba pokoleń nie ma znaczącego wpływu na najlepszy wynik algorytmu. Jednak daje to do zastanowienia - dzięki metodzie ruletki lepiej dopasowany wynik algorytmu otrzymuje większą szansę wystąpienia jako rodzic w następnym pokoleniu, lecz idąc dalej krzyżówka PMX może sprawić, iż dwaj najlepiej dopasowani rodzice mogą wymienić swoje geny tak, że ich fenotyp potomków znacznie się pogorszy przez wymianę nieodpowiednich genotypów. Mutacja również może spowodować, iż potomek, który miał lepszy czas niż każdy z rodziców, jak i poprzednich potomków, pogorszy się poprzez wymianę nieodpowiednich genów.

Metoda turniejowa z wykorzystaniem krzyżowania operatorem losowym z uzupełnianiem
Dla 50 zadań :

3 pokolenia	5 pokoleń	10 pokoleń
Wynik: 3502	Wynik: 3502	Wynik: 3502

Dla 100 zadań:

3 pokolenia	5 pokoleń	10 pokoleń
Wynik: 7418	Wynik: 7418	Wynik: 7325

Dla 200 zadań:

3 pokolenia	5 pokoleń	10 pokoleń
Wynik: 13511	Wynik: 13511	Wynik: 13471

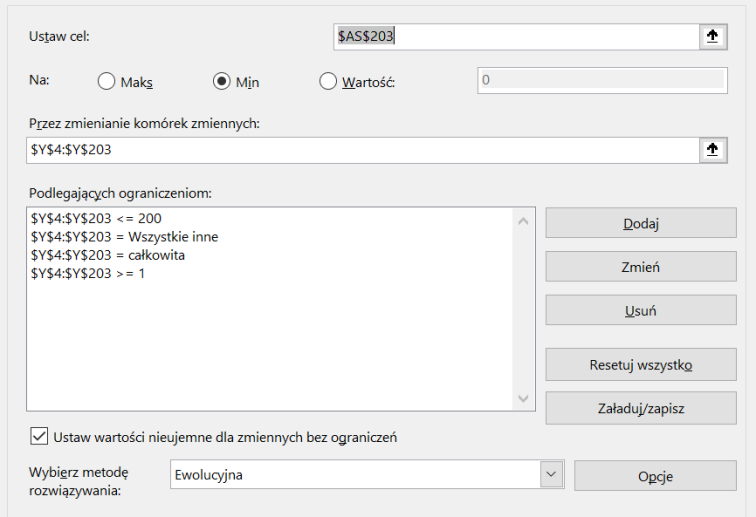
Wyniki odpowiadają za najlepszego dotychczas stworzonego potomka. W powyższych tabelach można zauważyć niewielką poprawę wyniku najlepszego fenotypu, ale dopiero przy 10 pokoleniu i to dla większej ilości zadań. Metoda turniejowa losuje dwóch osobników z populacji, a następnie porównuje ich funkcję dopasowania, dzięki czemu zawsze będzie wybrany lepiej dopasowany rodzic do stworzenia potomków, lecz ich krzyżowanie, jak i mutacja, może znacznie pogorszyć wynik końcowy algorytmu, tak jak to ma miejsce w metodzie ruletki.

Podaną hipotezę można uznać za prawidłową dla metody turniejowej z wykorzystaniem krzyżowania operatorem losowym z uzupełnianiem, lecz trzeba ją odrzucić dla metody ruletki i krzyżówką PMX. Podczas tworzenia algorytmu genetycznego można by usprawnić jego działanie poprzez sprawdzanie, czy funkcja dopasowania jest bardziej odpowiednia dla pary rodziców niż pary potomków, a następnie przypisanie lepiej dopasowanej funkcji jako kolejnego potomka, co pozwoliłoby na ciągłe polepszanie fenotypów osobników, a więc uniknięcie uzyskania w wyniku mutacji czy krzyżówki gorszego potomka. Jednak brak czasu oraz długie działanie algorytmu (2,5h) nie pozwolił na tak szczegółową analizę.

6. Solver

Dla porównaniu wyników użyto dodatku do programu Ms Excel – Solver. Umożliwia on przeprowadzanie analiz warunkowych. Za pomocą dodatku Solver można znaleźć optymalną (maksymalną lub minimalną) wartość formuły w jednej komórce — zwanej komórką celu — podlegającej ograniczeniom, czyli limitom. Dodatek Solver pracuje z grupą komórek, zwanych zmiennymi decyzyjnymi lub po prostu komórkami zmiennych, które służą do obliczania formuły w komórkach celu i komórkach ograniczeń. Dodatek Solver dostosowuje wartości w komórkach zmiennych decyzyjnych tak, aby spełnić limity obejmujące komórki ograniczeń i uzyskać pożądany wynik w komórce celu.

Jako komórkę celu oznaczono komórkę z wynikiem zakończenia ostatniego zadania przez ostatnią maszynę, a jako komórki ze zmiennymi decyzyjnymi zostały oznaczone komórki z numerami zadań. Na poniższym obrazie widoczne są użyte ograniczenia dla danych z 200 zadaniami:



Wyniki Solvera przedstawia poniższa tabela:

50 zadań	100 zadań	200 zadań
Wynik: 3490	**Wynik: 6712**	**Wynik: 12091**

Wnioski

Analizując powyższe zestawienia, można zauważyć, że w metodzie wspinaczkowej oraz symulowanego wyżarzania znaczący wpływ na wynik ma liczba iteracji. Obie metody wykazują zależność wykładniczą tej wartości do wyniku końcowego, jednak metoda symulowanego wyżarzania charakteryzuje się pod tym względem większą zmiennością. W metodzie Tabu Search liczba iteracji

posiada mniej zauważalny wpływ – po wykonaniu kilku iteracji algorytm znalazł optymalny wynik, który potem nie ulegał zmianie. Co więcej, w rozwiązywanym problemie, długość listy Tabu nie miała żadnego znaczenia na wynik końcowy, ale już zmiana kolejności początkowych zadań tak. Algorytm NEH nie ma żadnych parametrów, które mogą wpływać na wynik algorytmu, ponieważ jest algorytmem deterministycznym. Jedynym „parametrem", który zmienia wartość wyniku w tym algorytmie jest liczba zadań/maszyn. Zaś w algorytmach genetycznych liczba potomków wpływa tylko na wynik w przypadku metody turniejowej, ale z wykorzystaniem krzyżowania operatorem losowym.

Porównując uzyskane wyniki do wartości otrzymanej dzięki dodatkowi Solver, metoda symulowanego wyżarzania przy ustalonych liczbach iteracji nie pozwoliła na uzyskanie lepszego wyniku. Metoda wspinaczkowa pozwala na uzyskanie zadawalającego wyniku – lepszego niż „zaproponowanego" przez dodatek Solver, jednak tylko w przypadku wykonania dużej liczby iteracji. Najefektywniejszą metaheurystyką okazała się metoda Tabu Search, dzięki której uzyskano wynik lepszy od dodatku Solver już po dwóch iteracjach i najlepszy wynik ze wszystkich metod po wykonaniu pięciu iteracji oraz w każdym przypadku zmiany kolejności zadań. Podsumowując – dla 50 zadań najlepsza metoda to Tabu Search z pozmienianą kolejnością zadań, dla 100 jest to wspinaczka bez multistartu, a dla 200 wspinaczka bez multistartu oraz wyżarzanie ze względu na liczbę iteracji. Poniższa tabela jest zestawieniem wyników, potwierdzająca powyższe konkluzje (wybrano najlepsze wyniki spośród trzech wyników ze zmianami danego parametru):

50 zadań:

Solver	**3490**
Wspinaczka	**3202**
	3202
Wyżarzanie	**3433**
	3212
Tabu	**3202**
	3202
	3142
NEH	**3591**
Genetyczny	**3536**
	3468

100 zadań:

Solver	**6712**
Wspinaczka	**6497**
	6676
Wyżarzanie	**7575**
	7269
Tabu	**6631**
	6793
	6615
NEH	**7938**
Genetyczny	**7412**
	7325

200 zadań:

Solver	**12091**
Wspinaczka	**11795**
	11933
Wyżarzanie	**11795**
	11933
Tabu	**11852**
	11960
	11852
NEH	**13783**
Genetyczny	**13443**
	13471

Z powyższych tabel wynika, że najlepszą metodą okazała się być metoda Wspinaczki – przy każdym zestawie zadań jest ona lepsza od Solvera. Bardzo podobne wyniki uzyskują metoda Wyżarzania oraz Tabu Search. Niestety najgorsze (największe) wyniki otrzymywane są przez algorytm NEH, a w drugiej kolejności przez algorytm Genetyczny.