Problem przydziału dla symulowanego wyżarzania

E. Krupa

Politechnika Wrocławska

244993@student.pwr.edu.pl

Streszczenie

Niniejsza praca stanowi wstęp w tematykę problemu przydziału oraz algorytmu symulowanego wyżarzania, oraz przedstawia kilka propozycji rozwiązania różnych wariantów powyższego problemu przy użyciu tego modejścia metaheurystycznego. Ponadto zawiera ogólne omówienia algorytmów, jak również przykład dla niewielkiego rozmiaru problemu, porównanie własności rozwiązań zaprezentowanych w cytowanych źródłach oraz wnioski wskazujące na główne trudności w rozważanym problemie, a ponadto rekomendacje odnośnie wartości parametrów, metod doboru rozwiązań początkowych czy kryteriów zakończenia.

Słowa kluczowe: Problem przydziału; Symulowane wyżarzanie; Algorytm Genetyczny; Job shop scheduling; Job-shop problem; Simulated annealing; Genetic algorithm; Multi-stage flow shop scheduling problem

1. Problem przydziału

Problem przydziału (job shop scheduling) to problem optymalizacyjny, który polega na przydziale zadań do wykonania do pracowników/maszyn/procesorów w taki sposób, aby ich wykonanie zajęło jak najmniej czasu. Podstawowa wersja tego problemu zakłada istnienie n zadań $\mathrm{J}_1, \mathrm{J}_2, \ldots, \mathrm{J}_n$ o różnych poziomach wymagań obliczeniowych do ich wykonania, oraz m procesorów takich samych procesorów. Każdy z procesorów może w danym momencie wykonywać tylko jedno zadanie. Celem jest zminimalizowanie czasu, który upłynie pomiędzy rozpoczęciem pracy, a zakończeniem wszystkich zadań. Bardziej skomplikowana wersja zakłada, że w każde zadanie J_i składa się z podzadań $\mathrm{P}_1, \mathrm{P}_2, \ldots, \mathrm{P}_n$, które muszą być przetwarzane na konkretnych maszynach, i tylko jedno z podzadań może być przetwarzane w danym momencie, a ponadto zadania muszą być wykonywane w konkretnej kolejności. Istnieje wiele innych wariantów tego problemu, m.in. takie, które dodają czas, jaki jest potrzebny maszynie na przełączenie się między zadaniami, czy sprawiają, że czasy działania stają się probabilistyczne, a nie z góry ustalone.

2. Symulowane wyżarzanie

Symulowane wyżarzanie (simulated annealing) to rodzaj algorytmu metaheurystycznego przeszukującego przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych. Zwykle używany jest w przeszukiwaniu przestrzeni dyskretnej (taka występuje w problemie przydziału, czy też, np. w problemie komiwojażera), oraz w problemach, gdzie znalezienie w konkretnym przedziale czasu przybliżonego globalnego optimum jest ważniejsze niż znalezienie dokładnego optimum lokalnego. Algorytm został zainspirowany procesem obróbki cieplnej metali, w którym stopniowo schładzany metal wraz ze spadkiem temperatury traci energię termodynamiczną. Symulowane wyżarzanie

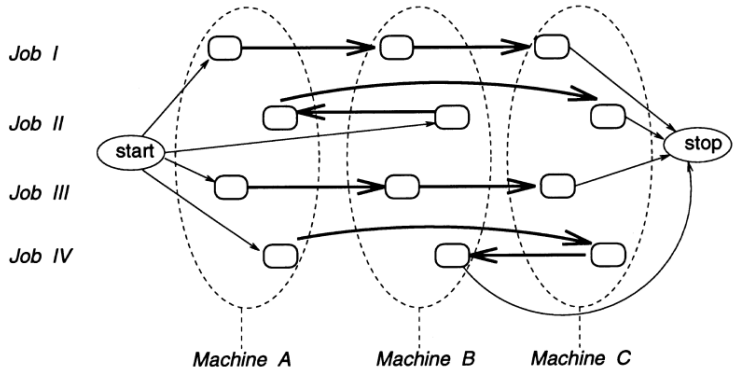
zakłada, że układ wraz ze spadkiem temperatury traci „chęć” do wybierania słabszych rozwiązań.

Rozwiązania wybierane są ze zbioru rozwiązań sąsiednich do aktualnego rozwiązania. Szansa przejścia do słabszego rozwiązania zależy od aktualnej temperatury, która spada, a tempo jej spadku zmniejsza się. Dzięki temu, na początku pracy algorytmu, jest on skłonny do wybierania różnych rozwiązań, zarówno lepszych jak i gorszych, w celu jak najszerszego przeszukania przestrzeni, w której mogą znajdować się optima lokalne. Z czasem jednak szansa na wybór słabszego rozwiązania maleje, przez co algorytm zawęża przestrzeń poszukiwań do lokalnych optimów, ostatecznie wybierając najlepsze rozwiązanie jakie był w stanie znaleźć.

Bardziej precyzyjnie mówiąc, zdefiniujmy temperaturę t jako wartość dodatnią. Niech S będzie skończoną przestrzenią rozwiązań, takich, że $c: S \to R_+$, gdzie c jest funkcją kosztu, którą minimalizujemy. Weźmy pewne obecnie rozważane (początkowe) rozwiązanie $x \in S$. Zdefiniujmy sąsiedztwo rozwiązań $N(x) \in S$, oraz weźmy jednego sąsiada y z $N(x)$. Zdefiniujmy również funkcję $\alpha(x, t, y)$, która przyjmuje wartość 1, gdy $c(y) < c(x)$, jednak w innych przypadkach opisana jest wzorem $e^{-\frac{c(y)-c(x)}{t}}$. Funkcja ta odpowiada za prawdopodobieństwo przejścia do danego sąsiada po jego wybraniu. Dla $c(y) < c(x)$ szansa ta wynosi 100%, więc y staje się zawsze nowym rozwiązaniem bieżącym (x := y). W przeciwnym wypadku y również może stać się aktualnym rozwiązaniem, jednak to prawdopodobieństwo jest niższe i zależne od aktualnej temperatury. Szansa na akceptację gorszego rozwiązania spada również wraz ze spadkiem jego jakości względem rozwiązania aktualnego. Algorytm zakończy się, kiedy zostaną spełnione pewne specyficzne kryteria (z reguły spadek temperatury do zdefiniowanego poziomu). Zdefiniujmy również $\pi_t(x) = \frac{e^{-\frac{c(x^*)-c(x)}{t}}}{\sum_{y \epsilon S} e^{-\frac{c(x^*)-c(x)}{t}}}$, jest to funkcja określająca prawdopodobieństwo wyboru danego sąsiada x, gdzie x^* jest stanem optymalnym.

3. Problem przydziału z podzadaniami [1]

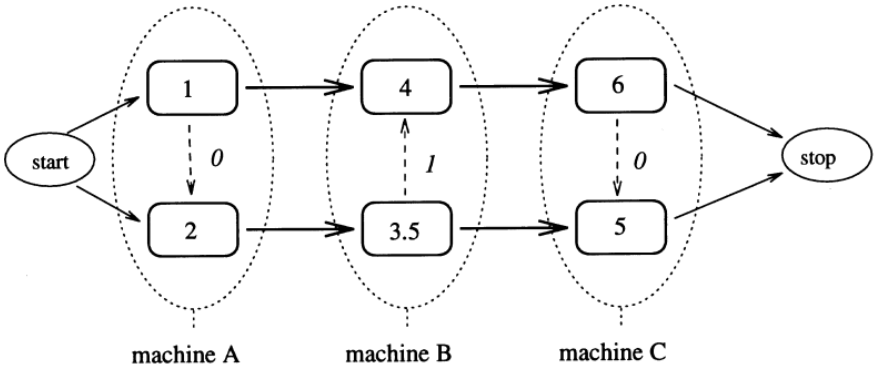
W tym rozdziale przeanalizujemy wariant problemu job shop scheduling, która zakłada istnienie n zadań $J_1, J_2, \dots, J_n$, m maszyn $M_1, M_2, \dots, M_m$, oraz m podzadań dla każdego zadania, z których każde musi być wykonane na innej maszynie.



Rysunek 1. Pochodzi z [1]. JSP z czterema zadaniami, każde z nich składa się z 3 podzadań, a każde z podzadań może być wykonane na jednej konkretnej maszynie.

3.1. Standardowy algorytm symulowanego wyżarzania

Standardowy sposób rozwiązania tego problemu za pomocą algorytmu symulowanego wyżarzania może okazać się nie wystarczający, z uwagi na to, że w pewnych konkretnych sytuacjach algorytm ma szansę wpaść w lokalne minimum, które będzie rozwiązaniem dalekim od pożądanego, a z którego nie będzie miał możliwości się wydostać. Aby to pokazać, rozważmy bardzo prosty przykład problemu JSP z dwoma zadaniami, z których każde składa się z trzech podzadań, pierwsze z nich wykonywane jest na maszynie A, drugie na maszynie B, a trzecie na maszynie C. Nietrudno policzyć, że istnieje 8 sposobów rozwiązania tego problemu (każda z trzech maszyn ma 2 zadania do wykonania, a więc $2!^3 = 8$).

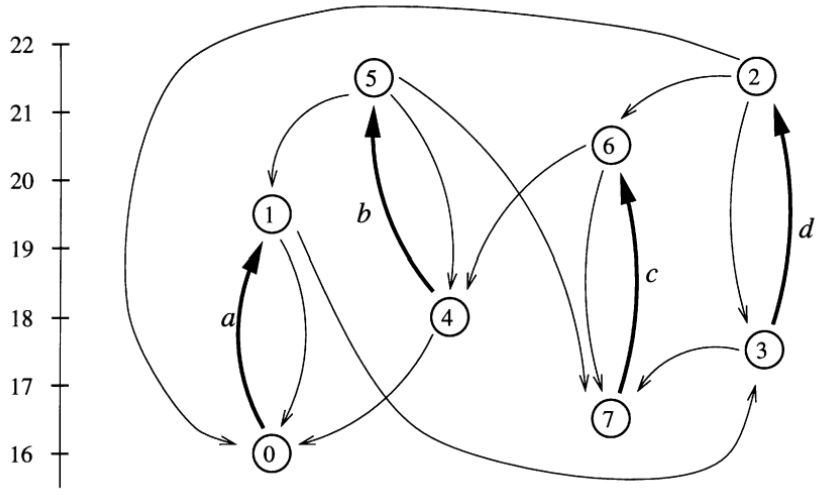


Rysunek 2. Pochodzi z [1]. Rozwiązania różnią się układem przerywanych strzałek. Każde z nich może być zwrócona w dwóch kierunkach, określając tym samem priorytet wykonywania zadań.

Zinterpretujmy strzałki jako liczby binarne, zamieńmy je na liczby całkowite i zapiszmy tak przedstawione rozwiązania do tabeli, wraz z funkcjami kosztu dla każdego rozwiązania.

Tabela 1. Wszystkie rozwiązania wraz z kosztami

Rozwiązanie	000	001	010	011	100	101	110	111
Numer	0	1	2	3	4	5	6	7
Koszt	**16**	**19.5**	**21.5**	**17.5**	**18**	**21.5**	**20.5**	**16.5**



Rysunek 3. Pochodzi z [1]. Graf przedstawiający przejścia pomiędzy rozwiązaniami w algorytmie SA. Skala po lewej przedstawia koszt rozwiązania.

Oczywiście, przejścia w dół grafu są możliwe za każdym razem, gdy takowe zostaną wylosowane, ponieważ prawdopodobieństwo $\alpha(x,t,y)$ ich wybrania jest równe 1 z uwagi na niższą funkcję kosztu sąsiada. Przejścia w górę (zaznaczone pogrubionymi strzałkami) mają niższe prawdopodobieństwo, odpowiednio $a = b = e^{\frac{-3.5}{t}}$, oraz $c = d = e^{\frac{-4}{t}}$. Policzmy teraz prawdopodobieństwa wylosowania następnych rozwiązań przy t dążącego do 0 za pomocą wzoru, $\pi_t(x) = \frac{\pi'_t(x)}{C_t}$, gdzie C_t to suma prawdopodobieństw.

$$(\pi'_t(0), \dots, \pi'_t(7)) = (3c(6+b+4d+bd), 2ac(3+b)(3+2d), 3\mathrm{acd}(3+\mathrm{b}), 6\mathrm{ac}(3 + \mathrm{b}), 6\mathrm{ac}(3+\mathrm{d}), 3\mathrm{abc}(3+\mathrm{d}), 2\mathrm{ac}(3+2\mathrm{b})(3+\mathrm{d}), 3\mathrm{a}(6+4\mathrm{b}+\mathrm{d}+\mathrm{bd}))$$

$$C_t = 18a + 12ab + 18c + 72ac + 3bc + 33abc + 3ad + 3abd + 12cd + 33acd + 3bcd + 14abcd$$

Wynika z tego, że granice dla t dążącego do 0 wynoszą $\lim_{t\to 0} \pi_t = (0,0,0,0,0,0,0,1)$, a więc w końcowych fazach działania algorytm będzie miał tendencję do zatrzymywania się w rozwiązaniu x_7, podczas gdy najbardziej optymalne jest rozwiązanie x_0.

3.2. Algorytm symulowanego wyżarzania wspierany algorytmem genetycznym

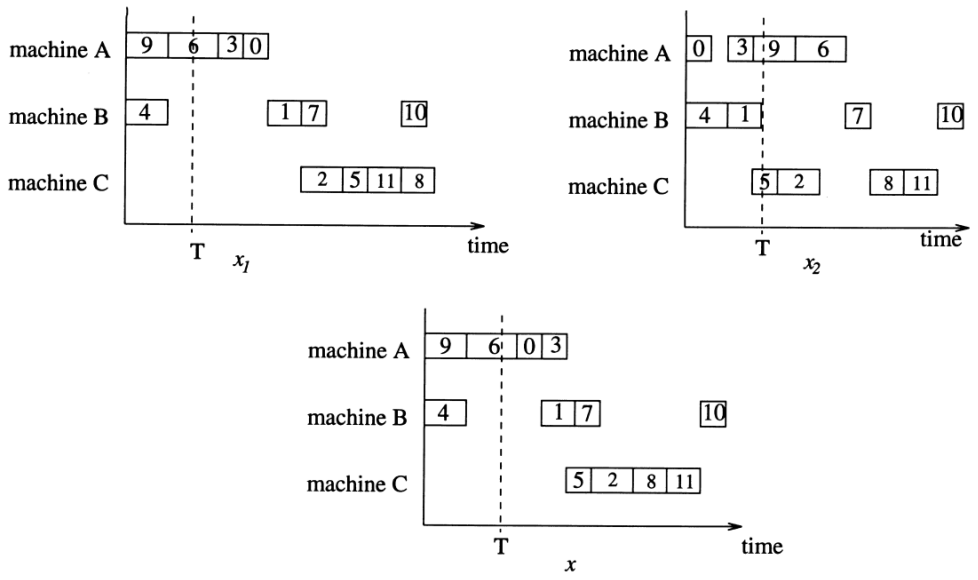
By sprawdzić, by algorytm nie blokował się w lokalnym minimum jak w powyższym przykładzie, spróbujemy połączyć go z algorytmem genetycznym (genetic algorithm), by był w stanie odwiedzić większą część przestrzeni rozwiązań przed zakończeniem działania. Pozwoli to nam również na zrównoleglenie obliczeń, z uwagi na to, że każdy osobnik może w danym momencie badać inną część przestrzeni rozwiązań.

3.2.1. Populacja

Populacja naszego algorytmu będzie składać się z K osobników, które będą reprezentować pojedyncze uruchomienia algorytmu symulowanego wyżarzania oraz będą zapamiętywać najlepsze znalezione przez siebie rozwiązania. Funkcja dostosowania, czy też zdolność adaptacji osobnika będzie oczywiście opisywana przez czas trwania tego najlepszego rozwiązania, a więc im niższa wartość, tym lepiej. W pierwszej generacji stworzone osobniki reprezentujące algorytmy symulowanego wyżarzania są inicjalizowane wartościami losowymi, z kolei w kolejnych iteracjach, losowo wybrane osobniki z poprzedniej iteracji będą krzyżowane w celu stworzenia nowych osobników, a gdy powstanie ich odpowiednia liczba, nadmiarowe zostaną zabite w celu utrzymania rozmiaru populacji K. Osobniki lepiej przystosowane będą miały większą szanse na reprodukcję. Szansa ta jest proporcjonalna do wartości wyrażenia we wzorze: $\max\{c(x') \mid x' \in \text{present population}\} - c(x)$.

3.2.2. Krzyżowanie, reprodukcja

Reprezentacje najlepszego rozwiązania znalezionego przez osobnika będziemy przedstawiać na wykresach jak niżej. Każdy rząd reprezentuje jedną maszynę oraz rozkład jej zadań w czasie (oś pozioma). Na początku wybieranych jest dwóch rodziców z rozkładem prawdopodobieństwa opisanym powyżej. Potem losowany jest moment w czasie pomiędzy 0 a momentem zakończenia pracy przez maszyny pierwszego rodzica. Następnie wszystkie operacje, które zaczynają się przed tym wylosowanym momentem lub dokładnie w tym momencie są przepisywane do nowego osobnika. Na koniec, wszystkie operacje maszyn drugiego rodzica, których jeszcze nie ma w nowym osobniku są przepisywane. Takie podejście zapewnia, że nowy osobnik będzie poprawną instancją algorytmu symulowanego wyżarzania, a schemat pracy maszyn będzie stanowił jako stan początkowy.



Rysunek 4. Pochodzi z [1]. Schematy krzyżowania się osobników w celu stworzenia nowego.

3.2.3. Kontrola temperatury

By umożliwić osobnikom łatwiejsze opuszczanie lokalnych optimów, wprowadzimy pojęcie kontroli temperatury. Oznacza to, że temperatura nie będzie już tylko spadać, ale będzie również mogła wzrosnąć na podstawie przebiegu procesu szukania rozwiązania. Bierzemy teraz pod uwagę całą sąsiedztwo ze stuprocentową szansą na przejście do niego po wyborze. Dla każdego z nich liczymy $d(x,y) \coloneqq c(x) - c(y)$. Weźmy d_{97}, czyli wartości wyższe niż 97% wybranych wartości, ale niższe niż pozostałe 3%. To oznacza, że zdecydowana większość wartości $\mathrm{d(x,y)}$ będzie się mieścić w przedziale $(-d_{97}, d_{97})$. Niech $d'(x,y)$ oznacza „znormalizowaną" różnicę kosztów, która przyjmuje wartości niemal tylko z przedziału $(-1,1)$, tj. $d'(x,y) = \frac{d(x,y)}{d_{97}} = \frac{c(x)-c(y)}{d_{97}}$. Weźmy też $v_0 > 0$, jako pewną małą wartość temperatury i większe od niej t_0 jako temperaturę początkową, oraz stała $0.5 < \delta < 1$. To obliczania temperatury w kolejnych iteracjach potrzebujemy jeszcze wzoru $\Delta_n \coloneqq \frac{t_0 - v_0}{n^\gamma}$, którego użyjemy do policzenia nowej temperatury $t' \coloneqq \max\{v_0, t - d_{n+1} \times \Delta_n\}$. Dzięki temu, temperatura jest w pełni kontrolowana przez algorytm i jeśli uda się znaleźć lepsze

rozwiązanie temperatura zmaleje i zawęzi nieznacznie obszar przeszukiwań, pozwalając się skupić na wyborze najlepszych wartości, natomiast w przeciwnym wypadku, kiedy znalezione rozwiązania będzie gorsze i zostanie zaakceptowane, temperatura wzrośnie zgodnie ze wzorem $\Delta\left(\frac{1}{\alpha(x,t,y)}\right) - 1 - \frac{c(x)-c(y)}{d_{97}}$ rozszerzając obszar poszukiwań i pozwalając na wyjście z lokalnego minimum. W przypadku, gdy gorsze rozwiązanie nie zostanie zaakceptowane, temperatura wzrośnie według wzoru $(1 + d'(x, y))^{+}\Delta_n$.

3.2.4. Rozwiązania początkowe, kryteria zakończenia

Rozwiązania początkowe można skonstruować potencjalnie na dwa sposoby. Pierwszy z nich to wybranie kolejności operacji i przetwarzanie operacji na każdej maszynie w odpowiedniej kolejności. Zapewnia ona jednak bardzo słaby start i jest ogólnie gorsza od drugiej metody, w której operacje są brane w losowej kolejności, a następnie w miejscu zajętym przez podzadania należące do danego zadania, podzadania są ustawiane zgodnie z wymaganym pierwszeństwem przetwarzania.

Kryterium zakończenia dla danego osobnika, czyli pojedynczej instancji algorytmu symulowanego wyżarzania to liczba l prób przejść do sąsiadów bez poprawienia najlepszego jak dotąd wyniku znalezionego w całym przebiegu algorytmu. Podobnie, algorytm genetyczny zatrzymuje się, kiedy w ciągu pewnej podanej liczby generacji g nie znajdzie osobnika osiągającego lepszy wynik, niż najlepszy do tej pory.

3.2.5. Parametry i ocena

Jeśli za parametry przyjmiemy l = 2 000 000 i g = 2, najlepsze rozwiązanie zostanie znalezione już po około 2/3 całego czasu trwania algorytmów. Dla przykładu, przy 20 zadaniach, 15 maszynach, problemie abz07 i najbardziej optymalnym rozwiązaniu równym 656, najlepszym znalezionym wynikiem w pięciu uruchomieniach był wynik 658, ze średnią

659,6. Z kolei dla takiego samego g, ale l = 10 000, rozwiązując problem law16 z 10 zadaniami i 10 maszynami, algorytm w pięciu podejściach przynajmniej raz znalazł najbardziej optymalne rozwiązanie równe 945, ze średnią wyników 945.2. Test dla większej ilości maszyn zajął jednak niemal 8h, a ten drugi tylko 38.8 sekundy, przy testach przeprowadzanych na procesorach odpowiednio Pentium 166 oraz Pentium 120. Różne instancje problemu mogą mięć wpływ na długość działania algorytmu, ale zdaje się, że zbyt wysoki parametr l jest szkodliwy i spowalnia algorytm powodując zbyt dużą ilość zbędnych prób, które nie są w stanie przynieść rozwiązania lepszego, niż te znajdowane już przy zdecydowanie niższym współczynniku l.

Jednakże po sparametryzowaniu też ilości osobników w danej generacji algorytmu genetycznego, testy wskazują, że najbardziej optymalnymi konfiguracjami są takie, w których ilość osobników utrzymuję się na niewielkim poziomie w stosunku do l. Dla przykładu, najlepszym pod względem znalezionego rozwiązania był test z ilością osobników równą 10, przy g = 1 i l =100 000, jednocześnie zajął on drugie miejscie pod względem wydajności czasowej, osiągając w tej drugiej kategorii nawet 5 krotnie lepsze wyniki niż test z ilością osobników równą 10 000, oraz g = 1 i l = 100.

3.3. Porównanie

Z uwagi na brak skłonności do blokowania się w lokalnym optimum, algorytm symulowanego wyżarzania wzmacniany algorytmem genetycznym zdaje się być zdecydowanie lepszą opcją od rozważanego w 3.1 podejścia standardowego. Wadą na pewno jest jednak wyższy stopień skomplikowania, co może przekładać się na błędy i wydłużać czas pisania kodu. W prostym problemach lub takich, gdzie lokalne optima są jak najbardziej odpowiednimi rozwiązaniami, należy się zastanowić nad sensownością implementacji tak skomplikowanego algorytmu.

4. Multi-stage flow shop scheduling problem [2]

W tym rozdziale będziemy rozpatrywać inny wariant problemu przydziału, multi-stage flow shop scheduling problem. Różni się on od problemu opisywanego w rozdziale trzecim tym, że podzadania nie muszą wykonywać się na konkretnych maszynach, jednakże dodano indywidualne dla każdego zadania czasy konfiguracji oraz czasy usunięcia.

4.1. Ulepszony algorytm symulowanego wyżarzania

4.1.1. Sąsiedztwo

Ta metoda stanowi odmienne podejście do algorytmu symulowanego wyżarzania, w uwagi na nową funkcję sąsiedztwa, która generuje sąsiadów na poniższe sposoby:

Niech n będzie liczbą etapów(procesów) harmonogramu:

1) Na pierwszym etapie (j = 1) rozkładu zadań s, pozyskujemy sąsiada przez zamianę zadań na dwóch pozycjach, po czym dalej zadania obsługujemy według kolejności przybycia (first come first service, FCFS) przez kolejne etapy (j = 2, 3, …, J).
2) Dla losowego etapu j != 1 pozyskujemy sąsiada przez zamianą zadań na dwóch pozycjach. Rozkłady zadań na pozycjach (j = j + 1, j + 2, …, J) obsługujemy zgodnie z FCFS.
3) W sposób losowy wybieramy opcję a lub b:
   a) Generujemy sąsiada tak jak w przypadku 1.
   b) Dla losowego etapu j != 1 pozyskujemy sąsiada poprzez zamianę zadań na pozycjach i oraz i`. i oraz i` można wyznaczyć z kolei na dwa sposoby:
      i) wybieramy je losowo, ale odpowiadających sobie zadań nie są przetwarzamy na tej samej maszynie,
      ii) sąsiadują ze sobą, odpowiadające sobie zadania na pozycjach i oraz i` przetwarzamy na tej samej maszynie i.

   Rozkład zadań na pozycjach (j = j + 1, j + 2, …, J) obsługujemy zgodnie z FCFS.

Zbiory sąsiadów mogą być generowane dla:

1) najlepszego rozkładu zadań dotychczas,
2) najlepszego rozkładu zadań na pewnym poziomie temperatury,
3) aktualnego rozkładu na pewnym poziomie temperatury,
4) dla losowego rozkładu na pewnym poziomie temperatury.

3 sposoby pozyskiwania sąsiadów oraz 4 ich „źródła" dają nam 12 sąsiadów na każdą iterację, najlepszy z nich zostanie wybrany dla dalszego rozwoju.

4.1.2. Rozwiązania początkowe, warunki obniżenia temperatury

By dobrać rozwiązanie początkowe należy dla każdego zadania zsumować czas stworzenia każdego z jego podzadań, a następnie uszeregować zadania rosnąco według obliczonej wartości. Dla dwóch pierwszych zadań zmieniamy status priorytetowy, by uzyskać dwie listy priorytetowe. Dla każdej z nich porządkujemy zadania, by zminimalizować całkowity czas działania maszyn i otrzymać listę którą wykorzystamy w następnej iteracji. Zadania są porządkowane według dwóch warunków.

1) Na pierwszym etapie, zgodnie z listą priorytetów, przypisujemy zadania do maszyny, która ma najniższy czas wykonania dla tego zadania.
2) Na pozostałych etapach zadania ustawiamy w kolejności zgodnie z FCFS. Zadania ponownie przydzielamy do maszyn o najniższym czasie ich przetwarzania.

Następnie dla zadań na pozycji i od 3 do n wybieramy zadanie na i-tej pozycji i umiejscawiamy je na wszystkich możliwych i-tych pozycjach listy priorytetowej, bez zmiany priorytetów zadań, które są na pozycjach od i + 1 do n. Na postawie listy i generujemy rozkłady prac, które spełniają wymagania naszego algorytmu. Odpowiednia lista priorytetowa, która wygenerowała najkrótszy czas pracy maszyn, zostanie użyta w kolejnej iteracji.

Podczas każdej iteracji algorytm decyduje, czy obniżyć temperaturę, czy też kontynuować szukanie optimum przy aktualnej temperaturze.

4.1.3. Parametry i ocena

Temperatura początkowa powinna zostać ustawiona w zakresie od 50 do 200, ze współczynnikiem redukcji na poziomie 0.95. Najbardziej optymalna temperatura minimalna wynosi 5.

Poniższa tabela przedstawia testy dla trzech różnych podejść do algorytmu przedstawionego w tym rozdziale:

1) A1- to zaproponowany algorytm wraz z zaproponowanymi rozwiązaniami początkowymi,
2) A2- to standardowy algorytm wraz z zaproponowanymi rozwiązaniami początkowymi,
3) A3- to zaproponowany algorytm z losowymi rozwiązaniami początkowymi.

Manfg. envirns.	Data sets	Performance of heuristics (in MRPD)			CPU times (s)		
		A1	A2	A3	A1	A2	A3
5 × 5	70	0.5334(46)	1.0219(32)	1.3718(27)	0.2563	0.1718	0.1963
5 × 15	70	0.2752(54)	0.7776(23)	0.8800(21)	2.8034	2.2466	2.7324
5 × 20	70	0.2504(53)	0.8489(25)	1.1075(27)	4.3895	4.0027	4.3310
15 × 5	70	0.7751(35)	1.3355(17)	1.4805(18)	2.6925	2.1121	2.6611
15 × 15	70	0.5338(32)	1.3648(19)	1.6270(19)	9.8924	9.6478	8.1548
15 × 20	70	0.5284(36)	4.2990(16)	5.2622(18)	15.1692	14.8288	14.1183
30 × 5	70	0.1935(57)	4.0222(8)	4.8222(5)	3.2005	2.5864	3.1017
30 × 15	70	0.0792(63)	3.3415(5)	4.5911(2)	14.3001	12.6457	12.9000
30 × 20	70	0.0949(61)	8.5361(5)	10.4496(4)	25.7778	23.2466	24.1956
50 × 5	70	0.0149(69)	7.1300(1)	10.3117(0)	5.3207	4.5942	5.0118
50 × 15	70	0.0000(70)	6.9543(0)	8.7180(0)	26.4504	23.1445	24.5832
50 × 20	70	0.0000(70)	9.1646(0)	11.8233(0)	41.8756	39.6687	40.2977
∑ MRPD		3.2788(646)	49.1463(146)	62.0950(146)			

Rysunek 5. Pochodzi z [2]. Porównanie efektywności i czasów wykonania trzech algorytmów.

Oczywiście, na pierwszy rzut oka widać, że algorytm A1 wypadł najlepiej w testach, osiągając wielokrotnie niższy czas przetwarzania problemu niż A2 i A3. Może wydawać się

zaskakujące, że A2 poradził sobie trochę lepiej niż A3. Wskazuje to na to, że w przypadku tych testów dobrze wybrane rozwiązanie początkowe jest bardziej istotne niż sam algorytm. Należy też zwrócić uwagę, że algorytm A1 wypadł najsłabiej pod względem czasu wykonania, ponieważ wykonywał się najdłużej, jednakże są to niewielkie różnice w porównaniu z konkurencją, i raczej nie istotne przy jego wydajności.

5. Podsumowanie, wnioski

Z uwagi na to, że w rozdziale trzecim i czwartym pracowaliśmy z różnymi wariantami problemu przydziału, ciężko jest porównać przytoczone algorytmy między sobą. Wynika to z faktu, że mają zastosowanie w innych sytuacjach. Gdyby jednak wziąć pod uwagę, jak wyróżniają się na tle standardowego algorytmu symulowanego wyżarzania, to obserwując wyniki testów z końców rozdziałów można dojść do wniosku, że algorytm z rozdziału czwartego poradził sobie lepiej, ponieważ znajdował rozwiązania kilkukrotnie wydajniejsze od algorytmów, z którymi był porównywany, mimo tego, że one same też były ulepszonymi wersjami standardowego algorytmu. Z kolei algorytm z rozdziału trzeciego powstał jako odpowiedź na wpadanie standardowego algorytmu w optima lokalne, co samo w sobie nie zawsze jest problematyczne. Z tego powodu uważam, że algorytm zaprezentowany dla problemu Multi-stage flow shop scheduling problem ma większą wartość użytkową.

Referencje

[1] M. Kolonko, Some new results on simulated annealing applied to the job shop scheduling problem, (1997), European Journal of Operational Research 113 (1999) 123-136

[2] Chinyao Low, Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines, Computers & Operations Research 32 (2005) 2013 – 2025