RÓWNOLEGŁY ALGORYTM SA DLA PEWNEGO PROBLEMU WIELOKRYTERIALNEGO

Jarosław PEMPERA, Dominik ŻELAZNY

Streszczenie: W pracy proponuje się nową koncepcję konstruowania algorytmów opartych na przeszukiwaniu przestrzeni rozwiązań, w którym istotnym elementem jest coraz powszechniej stosowane przetwarzanie równoległe. W pracy rozważany jest dwukryterialny problem szeregowania zadań w przepływowym systemie produkcyjnym, w którym każde stanowisko wyposażone jest w pojedynczą maszynę. Rozważane jest kryterium minimalizacji czasu zakończenia wszystkich zadań oraz kryterium minimalizacji sumy spóźnień realizacji zadań. Celem optymalizacji jest wyznaczenie zbioru Pareto optymalnych rozwiązań. Do rozwiązania problemu proponuje się nowy algorytm oparty na metodzie symulowanego wyżarzania. Elementy algorytmu wykorzystujące przetwarzanie równoległe oparte są na przetwarzaniu wektorowym, które jest dostępne praktycznie we wszystkich współcześnie produkowanych procesorach.

Słowa kluczowe: problem przepływowy, optymalizacja wielokryterialna, metoda SA.

1. Wprowadzenie

Postęp technologiczny nieodłącznie związany jest ze zwiększeniem wymagań klientów i koniecznością podnoszenia wydajności. Dotyczy to zarówno fabryk jak też narzędzi, którymi posługujemy się w codziennej pracy. Przedsiębiorstwa, aby utrzymać się na silnie konkurencyjnym rynku, zmuszone są do wykorzystywania coraz bardziej zaawansowanych systemów informatycznych wspomagających planowanie produkcji. Optymalizacja harmonogramowania produkcji opiera się na skomplikowanych modelach obliczeniowych. Coraz częściej obiektem optymalizacji są problemy wielokryterialne.

Dla zdecydowanej większości problemów optymalizacyjnych generowanych przez rzeczywiste systemy produkcyjne nie można skonstruować szybkich algorytmów dokładnych. Dlatego w praktyce stosowane są algorytmy heurystyczne oparte na metodach przeszukiwań przestrzeni rozwiązań. Zwiększenie jakości generowanych rozwiązań przez tego typu algorytmy pociąga za sobą zwiększenie czasu obliczeń i/lub konieczność stosowania coraz szybszych maszyn obliczeniowych. Jednakże kilka lat temu dotychczasowy schemat rozwoju procesorów uległ pewnej istotnej zmianie. Zwiększanie częstotliwości taktowania, które osiągnęło granicę możliwości technologicznych, zostało zastąpione przez zwiększanie liczby rdzeni i wątków w pojedynczym procesorze. Oprócz tego, zaprzęgnięto do obliczeń równoległych karty graficzne, wyposażone w wiele procesorów strumieniowych (*Stream Processors*). Zastosowanie metod równoległych pozwala na znaczne przyśpieszenie działania algorytmów w stosunku do ich odpowiedników sekwencyjnych oraz umożliwia tworzenie nowych metod konstruowania algorytmów w szczególności dla wymagających problemów wielokryterialnych.

1.1. Przegląd wybranych wielokryterialnych algorytmów szeregowania

Większość powszechnie używanych algorytmów optymalizacji wielokryterialnego szeregowania zadań wykorzystuje Pareto optymalność do oceny znalezionych rozwiązań. Najczęściej wykorzystywane w badaniach były algorytmy ewolucyjne oraz metody lokalnego przeszukiwania, jakkolwiek niewiele z zaproponowanych algorytmów zostało zaprojektowanych do działania równoległego.

Murata i inni [1] zaproponowali *Multi-Objective Genetic Algorithm* (MOGA), będący przedstawicielem algorytmów ewolucyjnych. Zaprojektowany został w celu rozwiązywania wielokryterialnego problemu przepływowego. Najistotniejszym elementem algorytmu jest zmodyfikowany operator selekcji, pozostałe elementy algorytmu są typowe dla algorytmów genetycznych dla problemów szeregowania. Selekcja związana jest ze zbiorem wag przypisanych do kryteriów optymalizacji, co pozwoliło na ukierunkowanie poszukiwania w kierunkach różnych kryteriów. W algorytmie zastosowano również mechanizm zachowywania elitarnych rozwiązań polegający na tym, że kilka z rozwiązań z frontu Pareto jest kopiowanych do kolejnej generacji. Murata i inni [2] zaprojektowali ulepszoną wersję algorytmu MOGA, nazwaną CMOGA. Najistotniejszą zmianą w nowym algorytmie była inna dystrybucja wag pomiędzy kryteriami optymalizacji. Wykorzystano strukturę komórkową, co pozwoliło na lepszą selekcję wag, co z kolei przyczyniło się do znajdywania lepszej aproksymacji frontu Pareto.

W swojej pracy, Chakravarthy i Rajendran [3] wykorzystują metodę lokalnego przeszukiwania. Ich celem była minimalizacja ważonej sumy dwóch kryteriów poprzez użycie prostego algorytmu symulowanego wyżarzania. Rozwiązanie początkowe wybierane jest spośród następujących metod: a) *Earlies Due Date* (EDD), b) *Least Static Slack* (LSS) oraz c) heurystyki NEH [4], podczas gdy generowanie sąsiedztwa wykonywane było za pomocą schematu zamiany sąsiadujących zadań.

Zainspirowani algorytmem *Pareto Archived Evolution Strategy* (PAES), Suresh i Mohanasundaram zaproponowali algorytm *Pareto Archived Simulated Annealing* (PASA) [5], który oparty był o nową metodę perturbacji. Mechanizm nazwany *Segment Random Insertion* (SRI) został użyty do generowania sąsiedztwa danego rozwiązania. W celu utrzymania niezdominowanych rozwiązań użyto zewnętrznego archiwum rozwiązań. Rozwiązanie początkowe generowane jest losowo, podczas gdy nowe rozwiązanie wybierane jest poprzez użycie skalowanej ważonej sumy kryteriów.

Odmianę algorytmu genetycznego, wyposażonego w procedurę inicjalizacji wstawiającą cztery dobre rozwiązania do początkowej losowej populacji, zaproponowali Pasupathy i inni [6]. Ich algorytm korzystał z zewnętrznej populacji do przechowywania rozwiązań niezdominowanych. Strategia ewolucji była podobna do tej użytej w algorytmu NSGA-II, podczas gdy poprawa jakości frontu Pareto bazuje na dwóch różnych procedurach lokalnego przeszukiwania, aplikowanych do zewnętrznej populacji po zakończeniu wykonywania głównej części algorytmu.

2. Zastosowanie przetwarzania równoległego

W ostatnich latach, obserwuje się dynamiczny rozwój urządzeń elektronicznych wykorzystujących procesory i układy realizujące przetwarzanie równoległe. Spowodowane jest to przede wszystkim dwoma czynnikami: (*i*) w praktyce osiągnięto górną taktowania cyfrowych układów elektronicznych, (*ii*) układy cyfrowe pracujące z wyższą częstotliwością zużywają znacznie więcej energii (zużycie energii jest proporcjonalne do

kwadratu częstotliwości). Wielordzeniowe procesory stosowane są powszechnie w komputerach klasy PC, komputerach przenośnych oraz coraz powszechniej w nowoczesnych telefonach komórkowych. Innymi elementami wykorzystującymi przetwarzanie równoległe są układy graficzne, układy przetwarzania dźwięku, szyfrowania pakietów danych.

Generalnie rozróżnia się dwa sposoby przetwarzania równoległego: przetwarzania drobnoziarniste i gruboziarniste. Przetwarzanie drobnoziarniste realizowane jest w jednościeżkowych algorytmach, w których pewne fragmenty kodu (najbardziej czasochłonne) realizowane są w sposób równoległy. Przetwarzanie gruboziarniste realizowane jest przez wiele wątków programu, z których każdy realizuje inną ścieżkę realizacji algorytmu. Ze względu na duże skomplikowanie kodu algorytmów, każdy wątek realizowany jest przez procesor, zatem realizacja przetwarzania gruboziarnistego możliwa jest w systemach zawierających procesory wielordzeniowe i/lub wiele komputerów połączonych siecią komunikacyjną.

Znacznie skromniejsze wymagania dotyczące sprzętu występują w przypadku przetwarzania drobnoziarnistego. Jedną z najbardziej najczęściej stosowanych metod równoległego przetwarzania drobnoziarnistego jest przetwarzanie wektorowe. Z chwilą pojawienia się procesorów Pentium z technologią MMX (ang. Matrix Math eXtensions) w końcówce lat 90. pojawiła się możliwość wykorzystania przetwarzania wektorowego realizowanego wewnątrz procesorów stosowanych w komputerach osobistych. Technologia SSE (ang. Streaming SIMD Extensions) następca technologii MMX oferuje przetwarzanie wektorowe realizowane na 128 bitowych rejestrach, w których mogą być zakodowane wektory składające się w zależności od rozmiaru elementu wektora od 2 do 16 elementów. Zestaw instrukcji wektorowych realizowany jest na dedykowanych rejestrach procesora i obejmuje on instrukcje arytmetyczne, logiczne, przesuwania oraz wymiany z pamięcią. Zastosowanie przetwarzania wektorowego potencjalnie pozwala na przyspieszenie obliczeń objętych przetwarzaniem od 2 do 16 razy.

3. Opis problemu

W przepływowym systemie produkcyjnym należy wykonać n zadań ze zbioru $J=\{1,\ldots,n\}$ na m maszynach ze zbioru $M=\{1,\ldots,m\}$. Każde zadnie $j\in J$ wykonywane jest dokładnie raz na każdej maszynie w kolejności zgodnej z numeracją zadań. Czas wykonania zadania j, $j\in J$ na maszynie k, $k\in M$ wynosi $p_{jk}>0$. Realizację zadania $j\in J$ należy zakończyć przed momentem d_j. Zadanie zakończone po tym terminie uważa się za spóźnione. Rozpoczętego zadania nie można przerywać aż do momentu zakończenia. W dowolnej chwili maszyna może wykonywać tylko jedne zadanie. Celem harmonogramowania zadań w systemie produkcyjnym jest określenie momentów rozpoczęcia i zakończenia wykonywania zadań na każdej maszynie. Przy czym muszą one spełniać opisane wyżej ograniczenia.

W ogólnym przypadku kolejność wykonywania zadań na maszynach możemy opisać przy pomocy zbioru składającego się m permutacji, z których każde składa się ze wszystkich elementów zbioru J. W pracy rozpatrujemy tzw. przypadek permutacyjny w którym zadania wykonywane są w tej samej kolejności na każdej maszynie. Zatem kolejność wykonywania zadań w systemie produkcyjnym możemy opisać przy pomocy permutacji zbioru $\{1,\ldots,n\}$.

Niech S_{jk} (C_{jk}) będzie momentem rozpoczęcia (zakończenia) wykonywania zadania j na maszynie k. Dla zadanej permutacji $\pi=(\pi(1),\ldots,\pi(n))$ określającej kolejność wykonywania zadań, momenty te muszą spełniać nierówności (1−4):

$$S_{jk}\geq 0,\quad j=1,\ldots n,\ k=1,\ldots m, \tag{1}$$

$$S_{\pi(j),k}\geq C_{\pi(j-1),k},\quad j=2,\ldots n,\ k=1,\ldots m, \tag{2}$$

$$S_{\pi(j),k}\geq C_{\pi(j),k-1},\quad j=1,\ldots n,\ k=2,\ldots m, \tag{3}$$

$$C_{jk}=S_{jk}+p_{jk},\quad j=1,\ldots n,\ k=1,\ldots m, \tag{4}$$

Momenty zakończenia spełniające ograniczenia (1)−(4) można wyznaczyć ze znanego wzoru rekurencyjnego (5), który można rozwiązać w sposób iteracyjny w czasie O(nm).

$$C_{\pi(j),k}=\max(C_{\pi(j-1),k},C_{\pi(j),k-1})+p_{\pi(j),k}, \tag{5}$$

gdzie $\pi(0)=0$, $C_{j,0}=0$ dla $j=1,\ldots,n$ oraz $C_{0,k}=0$ dla $k=1,\ldots,m$.

Zadanie optymalizacji polega na wyznaczeniu kolejności oraz harmonogramu wykonywania zadań minimalizującego następujące funkcje kryterialne:

1. moment zakończenia realizacji wszystkich zadań

$$C_{\max}(\pi)=\max_{1\leq j\leq n}\{C_{\pi(j)m}\}, \tag{6}$$

2. sumę czasów spóźnień

$$T_{tot}(\pi)=\sum_{j=1}^{n}T_{\pi(j)}, \tag{7}$$

gdzie $T_{\pi(j)}=\max(0,C_{\pi(j)}-d_{\pi(j)})$ jest spóźnieniem zadania $\pi(j)$. Wartości $C_{j,k}$ wyznaczone ze wzoru (5) są najmniejsze z możliwych, zatem minimalizują obie funkcje kryterialne dla zadanej kolejności wykonywania zadań π.

Celem optymalizacji jest wyznaczenie harmonogramu (permutacji) wykonywania zadań na maszynach minimalizującego jednocześnie obie funkcje kryterialne. Niestety w ogólnym przypadku rozwiązanie takie nie istnieje, zatem celem optymalizacji jest wyznaczenie zbioru permutacji optymalnych w sensie Pareto. tj. podzbioru zbioru wszystkich rozwiązań składającego się wyłącznie ze zbioru rozwiązań niezdominowanych .

W przypadku optymalizacji funkcji wielokryterialnej

$$F(\pi)=(f_1(\pi),\ldots,f_q(\pi)) \tag{8}$$

rozwiązanie α dominuje rozwiązanie π wtedy i tylko wtedy gdy

$$\text{dla każdego } k=1,\ldots,q,\quad f_k(\alpha)\leq f_k(\pi) \tag{9}$$

oraz

$$\text{istnieje } k=1,\ldots,q,\quad f_k(\alpha)<f_k(\pi). \tag{10}$$

Rozważany problem jest problem NP-trudnym, ponieważ zarówno problem przepływowy z minimalizacją czasu zakończenia realizacji zadań jak i problem przepływowy z minimalizacją sumy spóźnień są problemami NP-trudnymi.

4. Wyznaczenie harmonogramu dla wielu permutacji równolegle

W sekcji zostanie przedstawiony sposób wyznaczenia harmonogramu dla wielu permutacji przy wykorzystaniu przetwarzania wektorowego. Przetwarzanie wektorowe jest jednym z najprostszych w realizacji sprzętowej sposobów przetwarzania równoległego, niestety narzuca szereg ograniczeń, z których najistotniejszym jest dostęp do pamięci. Operandami operacji wektorowych są wektory zatem wszystkie dane wejściowe oraz wyjściowe muszą być przechowywane w elementach wektorów. Podobne ograniczenia dotyczą również innych sposobów przetwarzania drobnoziarnistego.

Przed przystąpieniem do omówienia metody wyznaczenie harmonogramu dla wielu permutacji oznaczamy przez $C_{[s]k}$ moment zakończenia wykonywania na maszynie k zadania stojącego na pozycji s w permutacji określającej kolejność wykonywania zadań. Dla zadanej permutacji π, równanie (5) przyjmuje postać (11)

$$C_{[s],k}=\max(C_{[s-1],k},C_{[s],k-1})+p_{\pi(s),k}. \tag{11}$$

Niech $\pi^{(1)},\pi^{(2)},\ldots,\pi^{(g)}$, będzie zestawem składającym się z g permutacji. Symbolem $C^{(x)}_{[s],k}$ oznaczamy $C_{[s],k}$ dla permutacji $\pi^{(x)}$, natomiast przez $p^{(x)}_{[s],k}=p_{\pi^{(x)}(s),k}$. Dalej, niech $\overline{C}_{[s],k}=(C^{(1)}_{[s],k},\ldots,C^{(g)}_{[s],k})$ oraz $\overline{p}_{[s],k}=(p^{(1)}_{[s],k},\ldots,p^{(g)}_{[s],k})$ będą wektorami zawierającymi w/w dane wejściowe i wyjściowe. Harmonogram wykonywania zadań dla g permutacji można wyznaczyć w czasie O(nm) wykonując obliczenia ma maszynie wektorowej przetwarzającej wektory składające się z g elementów realizującej kod przedstawiony na Rysunku 1.

```
1.    for s = 1 to n do
1.1     for k = 1 to m do
1.1.1     C̄[s],k = max(C̄[s−1],k , C̄[s],k−1) + p̄[s],k
```

Rys. 1. Pseudokod procedury wyznaczającej harmonogram dla g permutacji

Teoretyczne przyspieszenie obliczeń przy wykorzystaniu maszyny wektorowej wynosi g razy. Niestety, właściwe obliczenia muszą być poprzedzone zainicjowaniem elementów wektorów $\overline{p}_{[s],k}$, s=1,…,n, k=1,…,m. Inicjowanie to ze względu na zależność elementu od s, k oraz permutacji $\pi^{(x)}$, x=1,…,g może być zrealizowane sekwencyjnie w czasie O(gnm). Inicjację można jednakże znacząco przyspieszyć stosując odpowiednią organizację danych w pamięci oraz wykorzystując przetwarzanie wektorowe. W opisie metody przyjmuje uproszczenie (nie zmniejszające ogólności metody), że liczba maszyn m jest równa liczbie elementów wektora g.

Rozważmy krok 1.1 dla ustalonego s. Niech $T=[t_{xk}]_{m\times m}=[T_1,\ldots,T_m]$, gdzie $T_k=\overline{p}_{[s],k}$ będzie macierzą zawierającą czasy wykonywania zadań, które są potrzebne podczas wykonywania tego kroku. Zawartość macierzy T przedstawiona jest na Rysunku 2.

$p_{\pi^{(1)}(s),1}$	$p_{\pi^{(1)}(s),2}$	…	$p_{\pi^{(1)}(s),g}$
$p_{\pi^{(2)}(s),1}$	$p_{\pi^{(2)}(s),2}$	…	$p_{\pi^{(2)}(s),g}$
…	…	…	…
$p_{\pi^{(g)}(s),1}$	$p_{\pi^{(g)}(s),2}$	…	$p_{\pi^{(g)}(s),g}$

Rys. 2. Macierz T

Łatwo można zauważyć, że macierz T jest transpozycją macierzy T*, w której każda kolumna $T_x^*=(p_{\pi^{(x)}(s),1},p_{\pi^{(x)}(s),2},\ldots,p_{\pi^{(x)}(s),g})$ składa się z czasów wykonania tego samego zadania. Elementy wektora T_x^* można zainicjować w czasie O(1) wykorzystując operacje równoległego transferu z pamięci. Transpozycję macierzy T*, dla m będącego potęgą liczby 2 można wykonać w czasie O($m$log m) wykonując odpowiednią sekwencję przesunięć równoległych oraz zamian kolumn. Dla m=8 sekwencja przesunięć jest następująca:

sh1(T*[0],T*[1]) sh1(T*[2],T*[3]) sh1(T*[4],T*[5]) sh1(T*[6],T*[7])

sh2(T*[0],T*[2]) sh2(T*[1],T*[3]) sh2(T*[4],T*[6]) sh2(T*[5],T*[7])

sh4(T* [0],T*[4]) sh4(T*[1],T*[5]) sh4(T*[2],T*[6]) sh4(T*[3],T*[7])

gdzie

sh1(a,b): $a_1=a_1$, $a_2=b_1$, $a_3=a_3$, $a_4=b_3$, ..., $b_1=a_2$, $b_2=b_2$, $b_3=a_4$, $b_4=b_4$, ...,

sh2(a,b): $a_1=a_1$, $a_2=a_2$, $a_3=b_1$, $a_4=b_2$, ..., $b_1=a_3$, $b_2=a_4$, $b_3=b_3$, $b_4=b_4$, ..., itd.

Ostatecznie złożoność obliczeniowa procedury wyznaczenia harmonogramu dla g permutacji przy wykorzystaniu przetwarzania wektorowego jest rzędu O(nmlog g), zatem teoretyczne przyspieszenie obliczeń wynosi g/(log g).

5. Proponowany algorytm rozwiązania problemu

Zastosowanie algorytmów dokładnych do rozwiązania problemów NP-trudnych problemów harmonogramowania zadań produkcyjnych ogranicza się do rozwiązywania instancji o niewielkich rozmiarach (niewielkiej liczbie maszyn oraz zadań). Spowodowane jest to wykładniczym wzrostem czasu obliczeń wraz ze wzrostem rozmiaru problemu.

Z tego względu do rozwiązania tego typu problemów stosuje się algorytmy heurystyczne oparte na różnych metodach konstruowania. Do najbardziej znanych i najczęściej wykorzystywanych należą algorytmy oparte na metodach popraw. Spowodowane jest to takimi cechami jak: generowanie dobrej jakości rozwiązań końcowych, możliwość kompromisu pomiędzy czasem obliczeń (liczbą przeglądniętych rozwiązań) oraz jakością generowanego rozwiązania końcowego, stosunkowo łatwe zrównoleglenie w systemach wieloprocesorowych oraz prosta implementacja.

Jedną z najbardziej elastycznych pod względem adaptacji do różnych problemów optymalizacyjnych metod konstruowania algorytmów jest metoda symulowanego wyżarzania (SA ang. Simulated annealing). Metoda SA oparta jest na termodynamicznym procesie schładzania i została zaproponowana w pracy [7].

W przypadku optymalizacji jednokryterialnej funkcji kryterialnej $f()$, w każdej iteracji algorytmu opartego na metodzie SA, dla rozwiązania bieżącego x generowane jest rozwiązanie zaburzone x'. Jeżeli rozwiązanie x' nie jest gorsze od rozwiązania x to zastępuje je w następnej iteracji, w przeciwnym wypadku zastępuje je z pewnym prawdopodobieństwem. Prawdopodobieństwo akceptacji rozwiązania gorszego zmniejsza się wraz ze wzrostem różnicy wartości funkcji celu pomiędzy rozwiązaniem x' i x oraz zmniejsza się wraz z zmniejszaniem temperatury. Precyzyjniej, prawdopodobieństwo akceptacji rozwiązania gorszego wyraża się wzorem

$$P=\exp(-\Delta/T), \tag{11}$$

gdzie $\Delta=f(x')-f(x)$ oraz T jest temperaturą w danej iteracji.

Projekt algorytmu opartego na metodzie SA wymaga zdefiniowania postaci rozwiązania, sposobu generowania rozwiązania zaburzonego oraz wyboru lub zdefiniowania sposobu zmniejszania temperatury. Dodatkowo w przypadku optymalizacji wielokryterialnej należy podać zasady oceny rozwiązań w szczególności, kiedy rozwiązania nie jest gorsze od drugiego oraz w przypadku, gdy jest jak dużo.

W proponowanym algorytmie SA rozwiązanie reprezentowane jest w postaci permutacji, rozwiązanie zaburzone generowane jest przez losowy wybór rozwiązania z otoczenia typu *wstaw* (ang. *insert*). Realizowane jest to przez wygenerowanie dwóch liczb całkowitych a oraz b o rozkładzie jednostajnym z przedziału $[1,\ldots,n]$ oraz przesunięcie elementu $\pi(a)$ na pozycję b w π.

Bazując na propozycji przedstawionej w pracy [8], każde rozwiązanie π', które nie jest zdominowane przez rozwiązanie π traktowane jest jak rozwiązanie nie gorsze od π, natomiast każde rozwiązanie π' zdominowane przez π traktowane jest jak rozwiązanie gorsze od π, przy czym

$$\Delta = \sqrt{\sum_{i=1}^{q}(f_i(\pi') - f_i(\pi))}\ . \tag{12}$$

Podczas próbnych testów algorytmu SA zauważono, że algorytm generuje dobre rozwiązania, jeżeli rozpoczyna działanie w stosunkowo niskiej temperaturze wynoszącej ok. 100. Jednocześnie zaobserwowano, że podczas swojego działania znacząca frakcja 2/3 rozwiązań nie jest akceptowana. Narodziła się wówczas koncepcja wykorzystania obliczeń równoległych w nietypowy sposób, który będziemy nazywali obliczeniami w przód.

1. $t=t_0$, $\pi=\pi_0$, sel=0,
2. while ($t>t_k$)
2.1 if ($sel \geq g$) sel=0
2.2 if (sel=0)
2.2.1 generate $\pi'^{(1)},\ldots, \pi'^{(g)}$
2.2.2 parallel compute $f_1(\pi'^{(x)}), f_2(\pi'^{(x)})$, x=1,…,g
2.2.3 for x=1,…,g update Pareto set by pair $(f_1(\pi'^{(x)}), f_2(\pi'^{(x)}))$
2.3 $sel=sel+1$
2.4 if $\pi'^{(sel)}$ dominate π then $\pi=\pi'^{(sel)}$, sel=0
2.5 else compute Δ for $\pi'^{(sel)}$ and π, if $\pi'^{(sel)}$ is accepted then $\pi=\pi'^{(sel)}$, sel=0
2.6 $t=\lambda t$

Rys. 3. Schemat algorytmu SA z przetwarzaniem równoległym

W proponowanej koncepcji w każdej iteracji, w której nastąpiło zaakceptowanie nowego rozwiązania, generowanych jest g losowych permutacji sąsiednich oraz dla każdej z nich wyznaczany jest harmonogram wykonywania zadań oraz wartości funkcji kryterialnych. Następnie permutacje rozważane są pod względem akceptacji w kolejności $\pi'^{(1)},\ldots, \pi'^{(g)}$. Jeżeli permutacja $\pi'^{(x)}$, $1\leq x\leq g$ zostanie zaakceptowana to zastępuje permutację bieżącą i proces jest kontynuowany. Jeżeli nie zostanie zaakceptowana żadna z permutacji to generowane są kolejne g permutacje. Schemat proponowanego algorytmu został przedstawiony na Rysunku 3. Realizacją przetwarzania równoległego steruje zmienna *sel*. Zerowa wartość zmiennej *sel* oznacza wygenerowanie nowego zestawu permutacji oraz przeprowadzenie związanych z nim obliczeń równoległych (2.2.2). W kroku 2.2.3 aktualizowany jest zbiór Pareto−optymalny. Aktualizacja dotyczy wszystkich wygenerowanych permutacji. W krokach 2.4 oraz 2.5 przeprowadzony jest test akceptacji permutacji wskazywanej przez zmienną *sel*. W kroku 2.6 następuje obniżenie temperatury zgodnie z geometrycznym schematem schłodzenia.

6. Eksperyment komputerowy

Celem eksperymentu komputerowego było porównanie efektywności proponowanego algorytmu wykorzystującego przetwarzanie równoległe z efektywnością standardowego algorytmu. W języku C++ w środowisku Visual Studio 2005 zaimplementowano dwa algorytmy: SA – algorytm sekwencyjny oraz PSA – algorytm z wektorowym przetwarzaniem równoległym. Testy przeprowadzono na komputerze z procesorem Intel i7 2.4 GHz, przy czy obliczenia wykonano tylko na jednym rdzeniu. Testy algorytmów zostały przeprowadzone na zestawie instancji problemu przepływowego zaproponowanych przez Ruiza [9], które bazują na powszechnie znanych benchmarkach Tailard'a [10].

Tab. 1. Czas działania i przyspieszenie algorytmów

Grupa	CPU(SA)[s]	CPU(PSA)[s]	SLPR	SpeedUp0	SpeedUp1
20x5	0,10	0,20	2,63	0,50	1,31
20x10	0,20	0,41	3,45	0,49	1,68
20x20	0,29	0,65	3,94	0,45	1,76
50x5	0,19	0,39	2,45	0,49	1,20
50x10	0,31	0,67	2,47	0,46	1,14
50x20	0,60	1,02	2,76	0,59	1,62
100x5	0,24	0,62	2,13	0,39	0,82
100x10	0,55	1,22	2,46	0,45	1,11
100x20	1,10	1,98	2,66	0,56	1,48
200x10	0,92	2,28	2,30	0,40	0,93
200x20	2,00	3,80	2,70	0,53	1,42
Średnio			**2,7**	**0,48**	**1,31**

Poprzez sterowanie pracą generatora liczb losowych spowodowano, że algorytmy SA i SAR odtwarzały identyczną trajektorię przeszukiwań. Zatem algorytm PSA generuje rozwiązania co najmniej tak dobre jak algorytm SA. Ewentualne lepsze rozwiązania generowane przez PSA pochodzą ze zbioru rozwiązań dla których zostały wyznaczone wartości funkcji celu, ale nie zostały poddane procesowi akceptacji z powodu zaakceptowania wcześniej rozpatrywanej permutacji. Algorytmy SA i SAR dla każdej instancji zostały uruchomione 10 krotnie z rozwiązań wygenerowanych losowo, dla temperatury początkowej t_0=100 oraz końcowej t_k=1. Parametr λ został dobrany tak aby algorytmy dla zadanych temperatur wykonywały 10 000 iteracji. Przetwarzanie równoległe zrealizowano na wektorach o długości g=8.

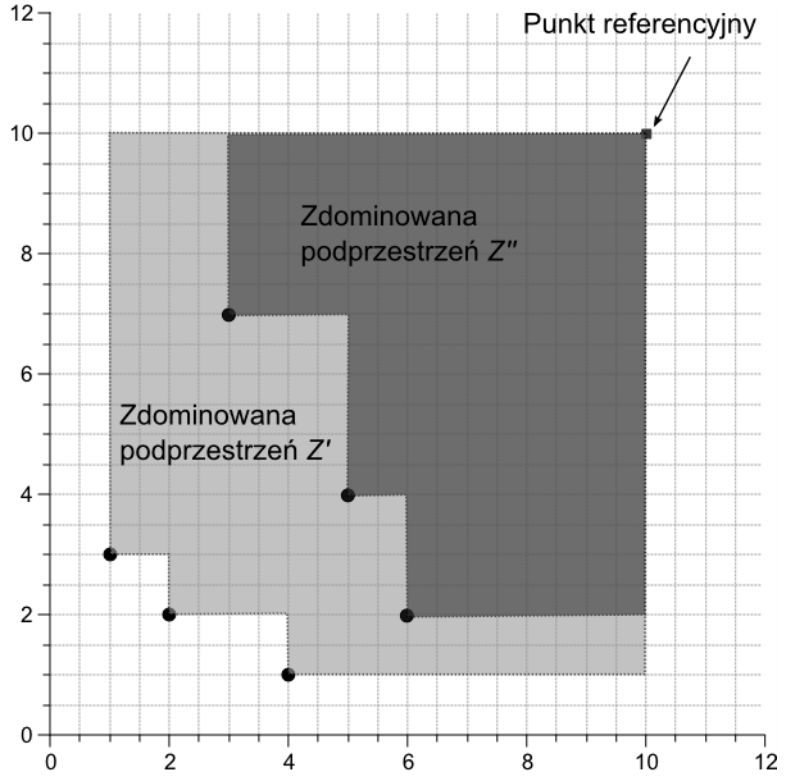
Podczas przebiegu eksperymentu komputerowego, dla każdej instancji i każdego algorytmu zapamiętano: zbiór rozwiązań niezdominowanych, czas obliczeń oraz w przypadku algorytmu PSA liczbę iteracji w których następowało przetwarzanie równoległe. W oparciu o zebrane w czasie eksperymentu dane wyznaczono wielkości charakteryzujące czas i przyspieszenie oraz jakość generowanych zbiorów rozwiązań niezdominowanych.

W tab. 1 przedstawiono wyniki badań eksperymentalnych mających na celu porównanie czasu działania algorytmów oraz przyspieszenia jakie uzyskuje się stosując przetwarzanie równoległe. W drugiej i trzeciej kolumnie przedstawiony jest rzeczywisty czas działania algorytmów, w czwartej stosunek liczby permutacji przetwarzanych przez PSA do liczby rozwiązań przetwarzanych przez SA (SLPR). W ostatnich dwóch kolumnach prezentowane jest przyspieszenie rzeczywiste SpeedUp0 = CPU(SA)/CPU(PSA) oraz przyspieszenie uwzględniające liczbę przeglądniętych permutacji przez algorytm PSA tj. SpeedUp1= SpeedUp0·SLPR.

Z rezultatów badań prezentowanych w Tab. 1 wynika, że algorytm PSA działa blisko dwukrotnie dłużej od algorytmu SA. Wyraźnie to widać analizując wartości współczynnika SpeedUp0. Jednocześnie łatwo można zauważyć, że algorytm PSA przegląda średnio 2,7

razy więcej permutacji niż algorytm SA. Biorąc pod uwagę ten fakt, przyspieszenie obliczeń mierzone stosunkiem liczby przetworzonych permutacji w ciągu jednostki czasu wynosi średnio 1.31. Przyspieszenie to wzrasta wraz ze wzrostem liczby maszyn i nieznacznie maleje wraz ze wzrostem liczby zadań.

Porównanie jakości rozwiązań generowanych przez algorytmy dla problemów wielokryterialnych wymaga zastosowania innych metod, niż proste porównanie wartości funkcji, z którym mamy do czynienia w przypadku jednego kryterium. W tym celu postanowiono wykorzystać wskaźnik hiper-objętości (*Hyper-Volume Indicator*) zaproponowany przez Zitzlera i Thiele w pracy [11] oraz opisany przez Knowlesa [12]. Porównanie jakości dostarczonych przez algorytmy aproksymacji frontu Pareto polega na wyznaczeniu punktu referencyjnego, najczęściej przyjmuje 120% najgorszych wartości dla poszczególnych kryteriów we wszystkich porównywanych zbiorach, oraz obliczeniu rozmiaru obszaru ograniczonego punktem referencyjnym oraz punktami reprezentującymi rozwiązania z frontu Pareto danego algorytmu. Przykład działania przedstawiono na rys. 4.



Rys. 4. Wizualizacja działania wskaźnika hiper-objętości

W tab. 2 przedstawiono wyniki porównania pokrycia zdominowanej podprzestrzeni rozwiązań algorytmu SA przez podprzestrzeń algorytmu PSA, liczbę rozwiązań w aproksymacjach frontu Pareto obu algorytmów oraz liczbę rozwiązań algorytmu SA zdominowanych przez rozwiązania algorytmu PSA. Jak łatwo zauważyć, w każdej z grup

instancji algorytm równoległy PSA odnalazł większe podprzestrzenie indykatora hiper-objętości niż algorytm SA.

Tab. 2. Sumaryczne porównanie jakości algorytmów

Grupa	Pareto (SA)	Pareto (PSA)	Zdominowane przez PSA	Zwiększenie pokrycia [%]
20x5	102	97	49	1,8
20x10	163	191	78	1,5
20x20	125	125	77	1,3
50x5	154	155	72	0,4
50x10	185	200	73	0,5
50x20	201	206	110	2,1
100x5	166	174	61	0,2
100x10	224	252	92	0,4
100x20	244	252	111	0,9
200x10	235	251	91	0,2
200x20	237	257	101	0,4
Średnio	**185**	**196**	**83**	**0,9**

Dodatkowo, blisko połowa rozwiązań znalezionych przez algorytm SA została zdominowana przez rozwiązania otrzymane w skutek działania algorytmu PSA. Powierzchnia podprzestrzeni indykatora była średnio o 0,9% większa dla algorytmu równoległo. W przypadku poszczególnych instancji algorytm równoległy uzyskał pokrycie nawet o ponad 5% większe niż algorytm sekwencyjny. Oba algorytmy nie różnią się konstrukcyjnie, a dzięki zrównolegleniu algorytm PSA w jednostce czasu przetwarza średnio 1,31 razy permutacji więcej. Można więc stwierdzić iż algorytm równoległy uzyskał nie tylko lepszą wydajność, ale też znajdywane przez niego rozwiązania dominowały rozwiązania algorytmu sekwencyjnego oraz uzyskał on lepsze pokrycie obszaru przestrzeni wartości funkcji kryterialnych.

7. Podsumowanie

W pracy zaproponowano nowy algorytm oparty na metodzie symulowanego wyżarzania dla dwukryterialnego problemu przepływowego. W algorytmie zastosowano wektorowe przetwarzanie równoległe, które zostało wykorzystane do wyznaczenia oryginalną procedurą harmonogramu dla wielu permutacji. Procedura ta została wykorzystana nowej propozycji modyfikacji metody symulowanego wyżarzania jakim są obliczenia w przód. Z rezultatów badań eksperymentalnych proponowanego algorytmu jednoznacznie wynika, że generuje on rozwiązania istotnie lepsze od rozwiązań generowanych przez klasyczny algorytm.

Literatura

1. Murata T., Ishibuchi H., Tanaka H.: Multi-objective genetic algorithm and its applications to flowshop scheduling, Computers and Industrial Engineering 30, 1996, 957–968.
2. Murata T., Ishibuchi H., Gen M.: Specification of genetic search directions in cellular multiobjective genetic algorithms, EMO '01 Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization, 2001, 82–95.
3. Charavarthy K., Rajendran C.: A heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization, Production Planning and Control 10, 1999, 707−714.
4. Nawaz M., Enscore Jr. E.E., Ham I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, OMEGA International Journal of Management Science 11, 1983, 91−95.
5. Suresh R.K., Mohanasundaram K.M.: Pareto archived simulated annealing for permutation flowshop scheduling with multiple objectives, IEEE Conference on Cybernetics and Intelligent Systems (CIS), 2004, 712−717.
6. Pasupathy T. Rajendran C., Suresh R.K.: A multi-objective genetic algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs, The International Journal of Advanced Manufacturing Technology 27, 2006, 804−815.
7. Kirkpatric S., Gelatt C., Vecchi M.: Optimisation by simulated annealing. Science 220, 1983, 671–680.
8. Pempera J. Smutnicki C., Żelazny D.: Optimizing bicriteria flow shop scheduling problem by simulated annealing algorithm, wysłane na konferencję: International Conference on Computational Science, Spain, 2013.
9. Minella G., Ruiz R., Ciavotta M.: A Review and Evaluation of Multiobjective Algorithms for the Flowshop Scheduling Problem, INFORMS Journal on Computing Summer vol. 20 no. 3, 2008, 451−471.
10. Taillard E.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64 (2), 1993, 278−285.
11. Zitzler E., Thiele L.: Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach, IEEE Transactiions on Evolutionary Computation 3(4) , 1999, 257−271.
12. Knowles J., Thiele L., Zitzler E.: A tutorial on the performance assessment of stochastic multiobjective optimizers., Tech. rep., ETH Zurich, 2006.

Dr inż. Jarosław Pempera
Mgr inż. Dominik Żelazny
Instytut Automatyki, Informatyki i Robotyki
Politechnika Wrocławska
50-372 Wrocław, ul. Wybrzeże Wyspiańskiego 27
tel./fax.: (71) 320-28-34
e-mail: jaroslaw.pempera@pwr.wroc.pl,
dominik.zelazny@pwr.wroc.pl