WIELOKRYTERIALNA OPTYMALIZACJA PRACY SYSTEMU WYTWARZANIA O STRUKTURZE PRZEPŁYWOWEJ

Dominik ŻELAZNY

Streszczenie: Praca dotyczy harmonogramowania zadań w sekwencyjnym przepływowym systemie produkcyjnym. W systemach wytwarzania o takiej strukturze różne produkty przechodzą przez tę samą ścieżkę technologiczną. Celem harmonogramowania w takim systemie jest wyznaczenie harmonogramu wykonywania zadań minimalizującego zadaną funkcję kryterialną. W opracowaniu rozważa się optymalizację dwukryterialną, na którą składają się niezwykle użyteczne funkcje jakimi są: najkrótszy czas zakończenia realizacji zadań oraz średni czas przepływu zadań przez system produkcyjny. W celu znalezienia przybliżonej granicy Pareto, zaproponowano nowy efektywny algorytm bazujący na Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II). Znaleziono również nowe właściwości problemu, które wykorzystano do skonstruowania i zmodyfikowania algorytmu NSGA-II. Przedstawiono również wyniki badań.

Słowa kluczowe: problem przepływowy, algorytm genetyczny, funkcja wielokryterialna, średni czas przepływu, długość uszeregowania.

1. Wstęp

Szeroko pojęta optymalizacja jest w dzisiejszych czasach nieodłącznym elementem sprawnie rozwijających się i działających na dużą skalę przedsiębiorstw oraz fabryk. Korzenie jednej z jej dziedzin sięgają aż początku XX wieku, gdy w latach 1908-1915 Henry Ford zaprojektował swoją pierwszą ruchomą linię montażową. Wynalazek ten doprowadził zarówno do przyśpieszenia produkcji jak i obniżenia jej kosztów, a co za tym idzie zrewolucjonizował wytwarzanie na masową skalę. Wraz z popularyzacją linii i taśm produkcyjnych i rozrostem skali produkcji, coraz trudniejsze stało się optymalne zaplanowanie wykonywania czynności produkcyjnych w tego typu systemach. Zaczęto więc szukać matematycznych rozwiązań, pozwalających na maksymalne wykorzystanie potencjału w nich drzemiącego. W ten oto sposób zapoczątkowana została nowa teoria matematyczna, zwana teorią szeregowania zadań.

W dzisiejszych czasach optymalne wykorzystanie czasu pracy jest istotnym aspektem rozwoju firm i przedsiębiorstw. W związku z tym szeregowanie zadań odgrywa istotną rolę w systemach wytwarzania przedsiębiorstw chcących utrzymać konkurencyjną pozycję na szybko zmieniającym się rynku, więc opracowywanie efektywnych i wydajnych zaawansowanych metod (lub algorytmów) szeregowania jest niezwykle istotne. Niniejsza praca dotyczy optymalnego harmonogramowania zadań przeznaczonych do wykonania w systemie produkcyjnym o szeregowej strukturze agregatów ciągu technologicznego. Jest to tak zwany problem przepływowy, który reprezentuje klasę szeroko badanych przypadków, bazujących na pomysłach pochodzących z inżynierii produkcji, która modeluje obecnie niemal jedną czwartą systemów produkcji, linii montażowych, informacyjnych obiektów usługowych [5]. Problem harmonogramowania zadań w systemie przepływowym jest z punktu widzenia złożoności obliczeniowej problemem NP trudnym

[2]. Większość z obecnie stosowanych kryteriów oceny jest łatwo adaptowalna do zastosowań rzeczywistych.

Znaczącym krokiem postępu w rozwoju teorii szeregowania zadań było opracowanie metody ścieżki krytycznej (CPM) w latach pięćdziesiątych. Jednymi z pierwszych, którzy zajmowali się problemem szeregowania, byli J.R. Jackson i E.R. Smith. Algorytm Jacksona został zaprojektowany w 1955 roku na potrzeby przemysłu, który zdaje się być głównym czynnikiem napędzającym rozwój wszelkich form optymalizacji. Od pionierskich prac z lat pięćdziesiątych zeszłego wieku, permutacyjny problem przepływowy (ang. Permutation Flow Shop Scheduling Problem – PFSSP) był wielokrotnie badany teoretycznie, matematycznie oraz empirycznie. Ze względu na jego złożoność, powstały metody podziału i ograniczeń oraz klasycznego programowania matematycznego [4], które dostarczają rozwiązań dokładnych, ale są użyteczne wyłącznie dla małych instancji problemów. W związku z tym powstało wiele algorytmów przybliżonych, włączając w to heurystyki konstrukcyjne, metaheurystyki popraw oraz algorytmy hybrydowe.

Próba rozwiązania wielokryterialnego problemu przepływowego jest naturalną ewolucja modeli oraz metod rozwiązania, zorientowanego na praktykę. W rzeczywistości, decyzje dotyczące szeregowania muszą brać pod uwagę kilka wskaźników ekonomicznych równocześnie.

W przeciągu ostatniej dekady zaproponowano szereg algorytmów rozwiązujących problemy wielokryterialne, głównie ze względu na ich zdolność do znajdywania w jednym przebiegu kilku rozwiązań optymalnych w sensie Pareto. Jako że nie jest możliwe, by jedno rozwiązanie było równocześnie optymalne dla wszystkich kryteriów, algorytmy dające rozwiązania znajdujące się bezpośrednio na lub w pobliżu frontu Pareto-optymalnego są niezwykle użyteczne w praktyce.

2. Opis problemu

Problem przepływowy składa się z dwóch głównych elementów, grupy m maszyn ze zbioru $M = \{1,2,\ldots,m\}$ oraz n zadań należących do zbioru $J = \{1,2,\ldots,n\}$, które zostaną przetworzone na tych maszynach. Każde zadanie musi przejść przez każdą maszynę, przy czym kolejność odwiedzania maszyn jest następująca: $1,2,\ldots,m$. Każde zadanie wykonywane jest tylko raz na jednej maszynie. Raz rozpoczęta operacja musi zostać wykonana do końca i zakończyć się przed operacjami, które poprzedza. W związku z tym zadanie $j, j \in J$, składa się z sekwencji m operacji. Zadanie j przetwarzane jest na maszynie k w nieprzerwanym czasie $p_{jk} > 0$.

Każda maszyna k, $k \in M$, może wykonać nie więcej niż jedno zadanie w tym samym czasie, każde zadanie może być wykonywane w jednej chwili tylko na jednej maszynie oraz zakłada się, że każda maszyna wykonuje zadania w tym samym porządku. Dozwolone uszeregowanie jest zdefiniowane przez czasy zakończenia C_{jk}, $j \in J$, $k \in M$ zadania $j$na maszynie k, spełniające powyższe ograniczenia. Dla zadanej kolejności wykonywania zadań reprezentowane permutacją $\pi = (\pi(1),\ldots,\pi(n))$ na zbiorze J, dozwolone uszeregowanie może być znalezione z pomocą poniższego wzoru rekurencyjnego:

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

gdzie: $\pi(0)=0$, $C_{j,0}=0$, $j \in J$, $C_{0,k}=0$, $k \in M$.

Funkcja oceny składa się z średniego czasu przepływu oraz całkowitej długości uszeregowania. Wartości w/w liczy się jak następuje:

$$C_{avg}(\pi) = \frac{1}{n}\sum_{j=1}^{n} C_{\pi(j),n} \text{ - średni czas przepływu,} \tag{2}$$

$$C_{\max}(\pi) = \max_{1\le s\le n} C_{\pi(s),m} = C_{\pi(n),m} \text{ - długość uszeregowania.} \tag{3}$$

3. Algorytmy przybliżone, problemy wielokryterialne

Wielokryterialny problem przepływowy badany był głównie w oparciu o algorytmy ewolucyjne. Stosowane metody to między innymi metoda ważonych celów, metakryterium, minimalizacja odległości od punktu idealnego oraz optymalność w sensie Pareto.

3.1. Początki optymalizacji wielokryterialnej

Algorytm VEGA (ang. Vector Evaluated Genetic Algorithm) został opisany w roku 1985 przez J.D. Schaffera [7]. Polega on na podziale populacji na *n*podpopulacji o jednakowych liczebnościach (*n* – liczba kryteriów). Każda podpopulacja poddawana jest niezależnej od innych selekcji, odpowiadającej innemu kryterium. Operacje mutacji i krzyżowania przeprowadzane są na całej populacji. Niewątpliwą zaletą jest łatwość implementacji, niestety algorytm ma tendencje do pomijania rozwiązań pośrednich, które dają dobre rozwiązania ze względu na wszystkie kryteria, ale nienajlepsze ze względu na każde z nich z osobna.

Algorytm HLGA (ang. Hajela's and Lin's Weighting-based Genetic Algorithm) opiera się o metodę celów ważonych. Oznacza to, że sprowadza zadanie wielowymiarowe (wielokryterialne) do zadania jednowymiarowego. Cel ten osiąga poprzez połączenie poszczególnych funkcji celu kryterialnych w jedną ważoną funkcję celu:

$$F(x) = \sum_{i=1}^{k} w_i f_i(x) \tag{4}$$

gdzie:

k – liczba kryteriów optymalizacji;
x – rozwiązanie;
$f_i(x)$ – i-ta funkcja kryterialna,
w_i – wagi takie, że:

$$w_i \in [0,1] \quad \text{oraz} \quad \sum_{i=1}^{k} w_i = 1$$

Różne wektory wag dają różne rozwiązania optymalne w sensie Pareto. Funkcję optymalizuje się przy użyciu opisanego już w niniejszej pracy algorytmu genetycznego. Podstawową wadą tej metody jest problem w doborze odpowiednich wartości wag dla poszczególnych kryteriów, co utrudnia uzyskanie dobrych jakościowo rozwiązań.

3.2. Algorytmy i metody obecnie używane

Ostatnie badania [9] jednoznacznie wykazują, że elitarność może znacząco zwiększyć wydajność algorytmów genetycznych. Dodatkowo pomaga zapobiec utracie dobrych rozwiązań, kiedy już się takie pojawią.Wśród elitarnych wielokryterialnych algorytmów ewolucyjnych (MOEA), najlepiej znane są algorytmy Strength Pareto Evolutionary Algorithm (SPEA) Zitzlera i Thiele'a [10], Pareto-archived Evolution Strategy (PAES) Knowlesa i Corne'a [3] oraz Elitist GA (EGA) Rudolpha [6].

Strength Pareto EA jest elitarnym wielokryterialnym algorytmem ewolucyjnym z koncepcją niezdominowania. Zasugerowano w nim utrzymywanie w każdej generacji zewnętrznej populacji zachowującej (kolekcjonującej) wszystkie poznane rozwiązania niezdominowane. Jest ona brana pod uwagę w trakcie operacji genetycznych, czyli krzyżowania i mutacji. Wszystkim rozwiązaniom niezdominowanym przypisywana jest wartość przystosowania, bazująca na liczbie rozwiązań przez nie zdominowanych, podczas gdy rozwiązaniom zdominowanym przypisywana jest wartość przystosowania gorsza od najgorszej wartości przystosowania rozwiązań niezdominowanych. W ten sposób kieruje się poszukiwania na rozwiązania niezdominowane. Dodatkowo wykorzystywana jest technika klasteryzacji mająca na celu zapewnienie zróżnicowania wyników.

W Pareto-archived ES, potomkowie porównywani są z rodzicami. Jeśli potomek dominuje rodzica, to ten ostatni zostaje odrzucony i zastąpiony w następnej iteracji swoim potomkiem. Gdy potomek jest zdominowany przez rodzica, zostaje odrzucony i nowy potomek zostaje wygenerowany. Gdy potomek i rodzic nie dominują się nawzajem, wtedy potomek porównywany jest z archiwum w celu sprawdzenia czy dominuje któregoś z członków archiwum niezdominowanych rozwiązań. Jeśli tak, potomek zostaje zaakceptowany jako nowy rodzic w następnej iteracji, a rozwiązania zdominowane zostają usunięte z archiwum. W przeciwnym wypadku, zarówno rodzic jak i potomek są porównywane pod względem bliskości do rozwiązań z archiwum i to, które zajmuje mniej zatłoczony obszar przestrzeni parametrycznej zostaje zaakceptowane jako rodzic i dodane do archiwum.

Rudolph zaproponował prosty elitarny wielokryterialny algorytm ewolucyjny, bazujący na systematycznym porównywaniu osobników z populacji rodziców i potomków. Rozwiązania niezdominowane z obu populacji są porównywane, by utworzyć nowy zbiór rozwiązań niezdominowanych, który staje się populacją rodziców w kolejnej iteracji algorytmu. Jeśli rozmiar tego zbioru jest mniejszy niż pożądany rozmiar populacji, uwzględniane są inne rozwiązania z populacji potomków. Niestety, algorytm ten nie utrzymuje zróżnicowania wyników optymalnych w sensie Pareto.

W pracy [1] Deb zaproponował algorytm Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II). Bazujący na algorytmie Non-dominited Sorting GA (NSGA), który krytykowano ze względu na wysoką złożoność obliczeniową niezdominowanego sortowania, brak elitarności i potrzebę sprecyzowana parametru współdzielenia, zmodyfikowano tak, by złagodzić powyższe problemy. Zastosowano szybkie sortowanie rozwiązań niezdominowanych, estymację gęstości i operator porównania zatłoczenia, które umożliwiły zmniejszenie złożoności obliczeniowej i poprowadzenie procesu selekcji w kierunku jednolicie rozpiętego frontu Pareto-optymalnego.

4. Zmodyfikowany algorytm NSGA-II

Liczne badania problemu przepływowego wykazują dość oczywistą tendencję, tj. rozwiązania o małej długości uszeregowania mają z reguły małą wartość średniego czasu

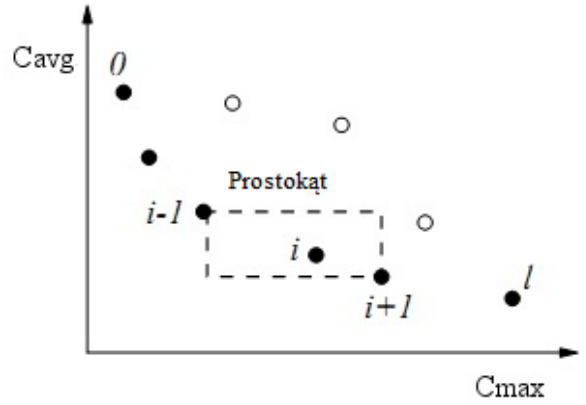
przepływu, natomiast rozwiązania dużej długości uszeregowania mają duży średni czas przepływu. Dodatkowo zaobserwowano, że dobre rozwiązania znajdują się zazwyczaj w okolicy innych dobrych rozwiązań dla problemu przepływowego z oboma kryteriami optymalizacji. Te właściwości wykorzystano w konstruowaniu modyfikacji algorytmu NSGA-II w celu zapewnienia większej efektywności.

Algorytm memetyczny Local Search Elitist Non-dominated Sorting GA (LS NSGA-II) wykorzystuje dwie techniki znane z oryginalnego algorytmu NSGA-II, mianowicie szybkie sortowanie rozwiązań niezdominowanych i estymację gęstości. Szybkie sortowanie rozwiązań niezdominowanych polega na tym, że wpierw dla każdego rozwiązania z populacji obliczane są dwa podmioty: (1)n_i, liczbę rozwiązań dominujących rozwiązanie$i$-te oraz (2)S_i, zestaw rozwiązań zdominowanych przez i-te rozwiązanie. Wszystkie rozwiązania, których współczynnik $n_i = 0$ dodaje się do listy F_1, która staje się obecnym frontem. Dla każdego rozwiązania i,$i = 1,\ldots,|F_1|$, w obecnym froncie przegląda się odpowiadającą mu listę S_ii zmniejsza o jeden współczynnik n_j każdego jej członka (j),j= 1,…,$|S_i|$. Jeśli, po wykonaniu przeglądu, któreś zj rozwiązań osiągnie wartość współczynnika $n_j = 0$, należy dodać je do osobnej listy H. Wszystkich zebranych członkówF_1 deklaruje się jako członków pierwszego frontu, a listę H nazywa obecnym frontem. Zabieg powtarzany jest, aż wszystkie zadania zostaną przydzielone do odpowiadających im frontom.

Algorytm szybkiego wyszukiwania rozwiązań niezdominowanych można zapisać w poniższym pseudokodzie:

```
fast-nondominated-sort(P)
    for each p∈ P
        for each q∈ P
            if (p ≺ q) then
                Sp = Sp∪ {q}
            else if (q≺p) then
                np = np + 1
        if np = 0 then
            F1 = F1∪ {p}
    i = 1
    while Fi ≠ ∅
        H =∅
        for each p∈ Fi
            for each q∈ Sp
                nq = nq – 1
                if nq = 0 then H = H∪{q}
    i = i +1
    Fi = H
```

Aby oszacować gęstość rozwiązań otaczających i-ty punkt, bierzemy średnią odległość dwóch najbliższych punktów otaczających i-ty punkt ze wszystkich stron ze względu na wszystkie kryteria. Wartość i_{distance} służy za estymator rozmiaru największego prostopadłościanu nie zawierającego innych niż i-ty punktów z frontu. Nazywa się to zatłoczeniem. Na rys. 1. przedstawiono zatłoczenie i-tego rozwiązania z jego frontu (zaznaczony wypełnionymi kropkami).



Rys. 1. Zatłoczenie punktu *i*-tego z frontu Pareto

Do policzenia zatłoczenia frontu F_j wykorzystywany jest poniższy algorytm:

```
crowding-distance-assignment(Fj)pochylić indeksy dolne
     l = |Fj|
     for each i, set Fj[i]distance = 0
     for each objective m
           Fj = sort(Fj, m)
           Fj[1]distance = Fj[l]distance = ∞
         for i = 2 to (l – 1)
                 Fj[i]distance = Fj[i]distance + (Fj[i+1].m - Fj[i-1].m)
```

gdzie $F_j[i].m$ odwołuje się do wartości *m*-tego kryterium *i*-tego rozwiązania w *j*-tym froncie.

W funkcji selekcji wykorzystywany jest operator porównania zatłoczenia ($\geq_n$), który kieruje w/w w różnych stadiach algorytmu, tak aby front Pareto-optymalny był jednolicie rozłożony. Każdy osobnik w populacji posiada dwa atrybuty: (1) stopień niezdominowania oraz (2) lokalny współczynnik zatłoczenia. Pomiędzy dwoma rozwiązaniami o różnych stopniach niezdominowania preferowane jest to o niższym froncie. W innym wypadku, gdy oba rozwiązania należą do jednego frontu, wybieramy to otoczone większym prostopadłościanem. Innymi słowy to, które znajduje się w obszarze mniej zatłoczonym.

Zmodyfikowany algorytm LS NSGA-II w każdej iteracji dokonuje dodatkowo ustalonej liczby iteracji stochastycznego przeszukiwania lokalnego (LS)dla każdego z osobników populacji potomnej, w celu polepszenia jakości rozwiązań potomnych. Lokalne przeszukiwanie obejmuje sąsiedztwo rozwiązań generowane przez przestawienia sąsiednie. Nowe rozwiązanie generowane jest przez losową zamianę dwóch przyległych zadań w obecnej permutacji. Jeśli dominuje stare rozwiązanie, to zastępuje je w kolejnej iteracji LS. W przeciwnym wypadku zostaje odrzucone.

Osobniki w populacji reprezentowane są przez permutację zadań, wartości funkcji kryterialnych, rangę Pareto oraz współczynnik gęstości. Algorytm LS NSGA-II wykorzystuje krzyżowanie o schemacie PMX oraz selekcję turniejową, jednocześnie

zachowując zewnętrzny zbiór rozwiązań niezdominowanych, aktualizowany w każdej iteracji. Dzięki takiemu podejściu algorytm może zachować do kolejnej iteracji rozwiązania zdominowane, które poprzez operatory genetyczne mogą prowadzić do lepszych rozwiązań niezdominowanych.

5. Wyniki obliczeń

Algorytmy NSGA-II oraz LS NSGA-II zaprogramowano w języku programowania C++ i skompilowano w Microsoft Visual Studio 2010, pracującym na komputerze PC z procesorem Intel Core i3-2100 3.1 GHz, 2x 4GB DDR3 1600MHz i systemie operacyjnym Windows 7 Profesional. Algorytmy były testowane na instancjach benchmarków dostarczonych przez Taillarda [8].

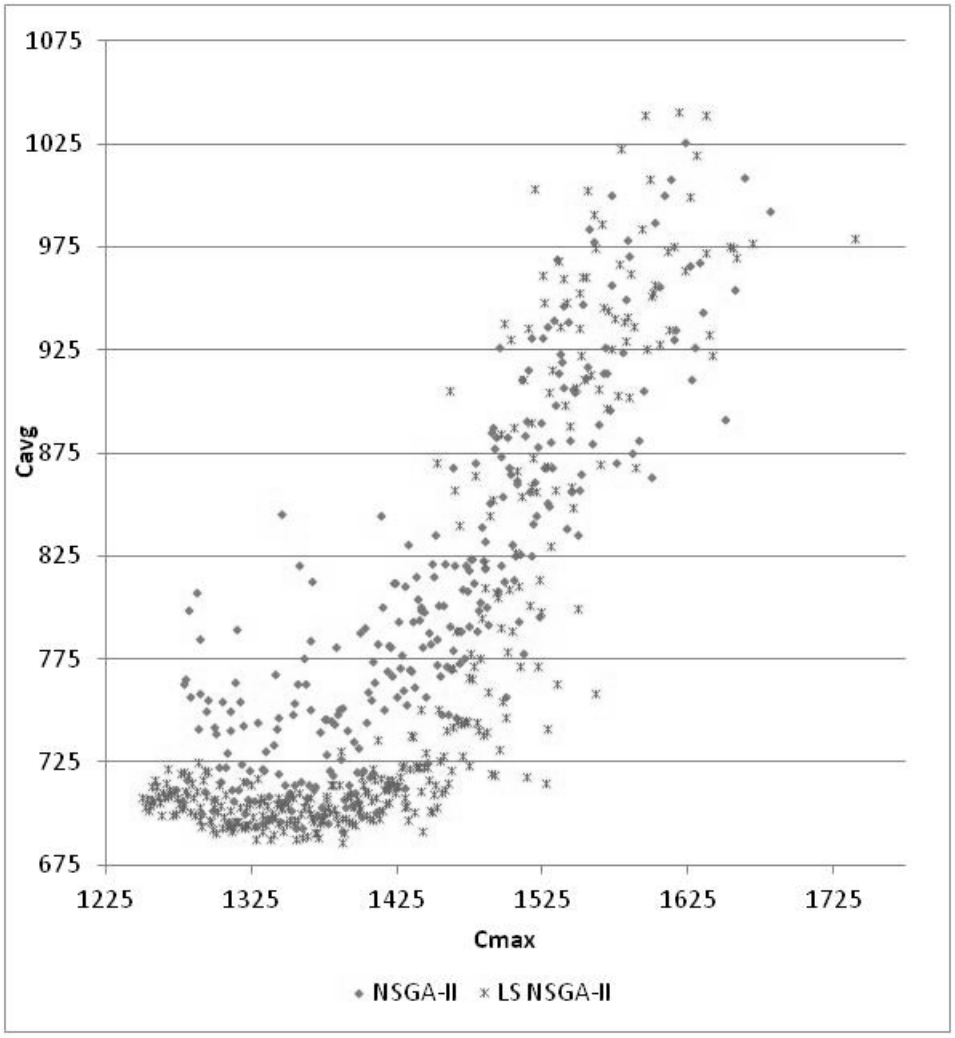
W pierwszym teście, obejmującym pięć instancji problemów, oba algorytmy wykonały 100,000 iteracji dla każdej z instancji. Dla każdego testowanej instancji zebrano zbiory rozwiązań Pareto-optymalnych P^A, A $\in$ {NSGA-II, LS NSGA-II}. Następnie ustalono zbiór P^* zawierający rozwiązania niezdominowane z obu zbiorów. Na koniec dla każdego algorytmu A ustalono liczbę rozwiązań d(A) z P^A, które znalazły się w P^*. Liczba niezdominowanych rozwiązań dla obu algorytmów, a także liczba elementów z każdego zbioru pokazane są w tabeli 1.

Tab. 1. Liczba rozwiązań Pareto-optymalnych

Instancja	NSGA-II		LS NSGA-II		$\|P^*\|$
	d	$\|P\|$	d	$\|P\|$	
TA05	0	6	10	10	10
TA25	0	4	6	6	6
TA35	0	3	7	7	7
TA41	3	7	8	8	11
TA60	0	3	6	6	6
TA61	0	2	3	3	3

Można zauważyć, że dla tej samej liczby iteracji niemal wszystkie rozwiązania znalezione przez algorytm NSGA-II zostały zdominowane przez rozwiązania zaproponowane przez algorytm LS NSGA-II. Dodatkowo, liczba rozwiązań Pareto-optymalnych znalezionych przez LS NSGA-II jest większa niż odpowiadająca jej liczba rozwiązań klasycznego algorytmu NSGA-II, co algorytm LS NSGA-II niewątpliwie zawdzięcza zastosowaniu metody lokalnego przeszukiwania.

Należy nadmienić, że z trzech rozwiązań znalezionych przez algorytm NSGA-II dla instancji TA41 aż dwa pokrywały się z rozwiązaniami znalezionymi przez algorytm LS NSGA-II. W związku z tym liczność zbioru P^*, będącego rozwiązaniami niezdominowanymi z sumy zbiorów rozwiązań obu algorytmów, jest w rzeczywistości mniejsza i wynosi 9. Nie licząc powtarzających się rozwiązań niezdominowanych algorytmy znalazły jedno i sześć, odpowiednio dla NSGA-II i LS NSGA-II, z rozwiązań niezdominowanych zebranych w P^*.



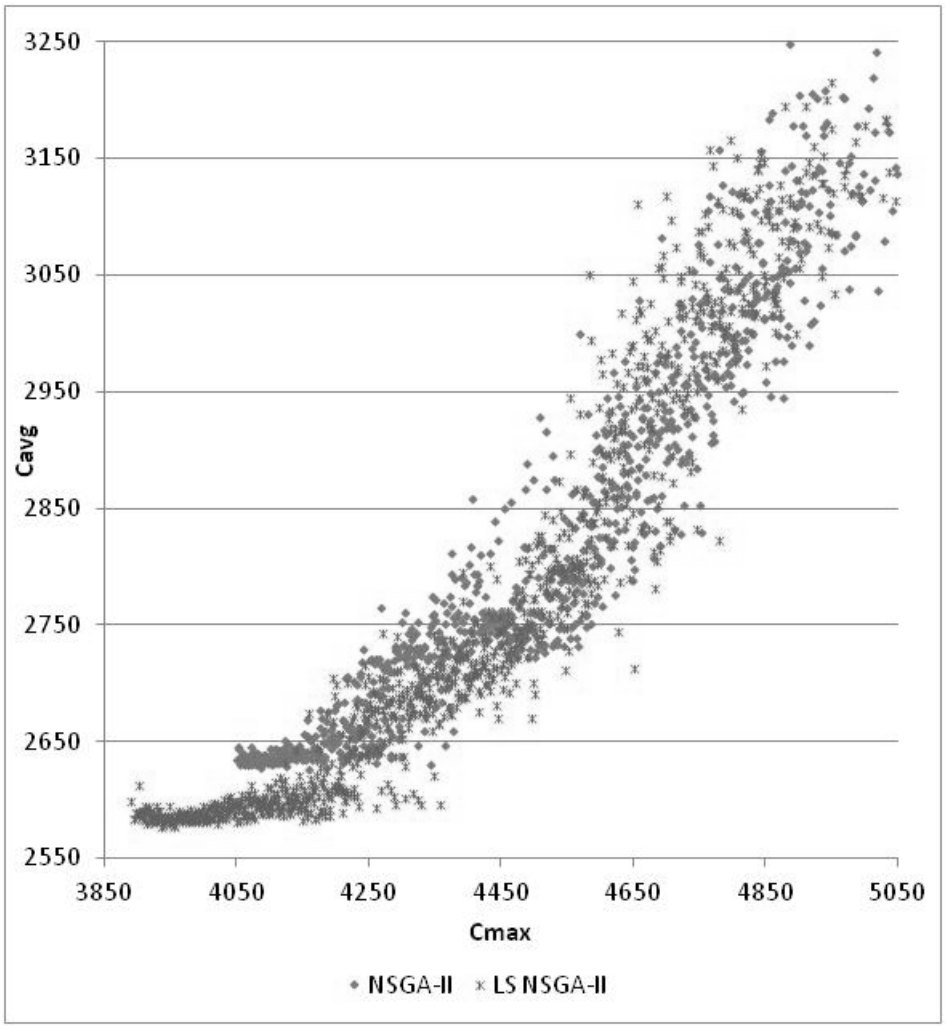
Rys. 2. Zbiór punktów „minimalnych” dla przypadku TA05

Dodatkowo, podczas badań zweryfikowano zależność obu funkcji od rozwiązań dla problemu przepływowego. W tym celu podczas działania algorytmu LS NSGA-II zachowano wartości obu funkcji dla wszystkich wygenerowanych rozwiązań. Niech $Z=\{(x,y)\}$ będzie zbiorem par wartości funkcji, gdzie x – oznacza długość uszeregowania, natomiast y – oznacza średni czas przepływu. Na podstawie zbioru stabelaryzowano funkcję

$$M(x) = \min_{(x,y)\in Z} y \tag{4}$$

Na wykresie rys. 2. przedstawiono wykres funkcji *M(x)*z przebiegu obu algorytmów dla instancji Taillarda TA05. Jak łatwo zauważyć, zbiór „minimów” algorytmu LS NSGA-II

osiągnął mniejsze wartości niż zbiór odpowiadający mu zbiór rozwiązań znalezionych przez algorytm NSGA-II. Jedynie niektóre z rozwiązań dostarczonych przez algorytm klasyczny znalazły się w pobliżu rozwiązań przedstawionych przez algorytm LS NSGA-II.



Rys. 3. Zbiór punktów „minimalnych” dla przypadku TA60

Zbiory podobne do tych z wykresu rys. 2. przedstawiono na wykresie rys. 3., w celu zobrazowania popraw, w stosunku do algorytmu pierwotnego, wynikających z działania algorytmu LS NSGA-II.

Warto również zauważyć, że wraz ze zwiększeniem rozmiaru instancji zwiększa się również różnica, na niekorzyść klasycznego NSGA-II, pomiędzy oboma algorytmami. Ma to zapewne związek z większą liczbą zadań i co za tym idzie większą liczbą dostępnych rozwiązań. Na zbadanych problemach, nie opublikowanych w postaci wykresów w

niniejszej pracy, można zaobserwować iż oba algorytmy posiadają zbliżone zbiory „minimów" dla małych wielkości instancji, natomiast różnice między nimi rosną wraz ze wzrostem liczby zadań.

Tab. 2. Procentowa liczba rozwiązań niezdominowanych

Rozmiar instancji	NSGA-II	LS NSGA-II
	d /|P*| [%]	d/|P*| [%]
20x5	20,55	79,45
20x10	30,49	69,51
20x20	34,26	65,74
50x5	3,80	96,20
50x10	20,51	79,49
50x20	21,67	78,33
100x5	34,09	65,91
100x10	40,00	60,00
100x20	21,99	78,01
200x10	4,76	95,24
200x20	0,00	100,00

Zbadano również zdolność obu algorytmów do znajdywania rozwiązań niezdominowanych w przypadku wprowadzenia dla obu tego samego ograniczenia czasu działania. W porównaniu do klasycznego algorytmu NSGA-II liczba iteracji algorytmu LS NSGA-II uległa zmniejszeniu.W trakcie testów zebrano liczbę rozwiązań niezdominowanych, które znalazły się w zbiorach P^*, a następnie policzono udział tych rozwiązań dla obu algorytmów. W tabeli 2 przedstawiono wyżej wymieniony w zależności od rozmiarów instancji. Można łatwo zaobserwować, że co najmniej 60,00% znalezionych rozwiązań Pareto-optymalnych należało do algorytmu LS NSGA-II. Podczas gdy algorytm klasyczny znalazł średnio jedynie 21,10% rozwiązań niezdominowanych.

6. Wnioski

W pracy [1] przedstawiono algorytm NSGA-II oraz efekty jego działania dla problemów ciągłych. Przystosowany do problemu przepływowego dał zadowalające wyniki, jednak przedstawiony w niniejszej pracy algorytm LS NSGA-II pozwolił na znalezienie rozwiązań, które w znacznej mierze zdominowały rozwiązania prezentowane przez algorytm klasyczny. Technika przeszukiwania lokalnego, zastosowana w celu zwiększenia efektywności, okazała się skuteczna pomimo zwiększonej złożoności obliczeniowej. Algorytm LS NSGA-II nawet w przypadku wykonania blisko pięciokrotnie mniejszej liczby iteracji znalazł znacznie więcej rozwiązań niezdominowanych.

Jak już wspomniano, algorytm memetyczny LS NSGA-II znajdywał rozwiązania Pareto-optymalne, które dominowały rozwiązania Pareto-optymalne znalezione przez algorytm NSGA-II. Dodatkowo, przebiegi funkcji $M(x)$wyznaczonej w trakcie działania algorytmu LS NSGA-II były położone bliżej osi wartości C_{max} niż analogiczne przebiegi funkcji $M(x)$ wyznaczone w trakcie działania algorytmu klasycznego. Różnice te

zwiększały się wraz ze zwiększeniem rozmiaru instancji. Jest to niewątpliwie zasługa zastosowania metody przeszukiwania lokalnego, która pozwoliła na znalezienie większej liczby rozwiązań i poprawienie już znalezionych.

W trakcie badań zaobserwowano również pewną właściwość problemu przepływowego z kryteriami maksymalnej długości uszeregowania i średniego czasu przepływu. Mianowicie, dobre rozwiązania mają tendencję do znajdywania się w okolicy innych dobrych rozwiązań, a wraz ze wzrostem wartości jednej funkcji zauważalnie rośnie wartości drugiej funkcji kryterialnej. Przedstawione na wykresach rys.2. oraz rys. 3. zależności obu funkcji doskonale obrazują powyższe. W ogonie funkcji znajdują się rozwiązania dalekie od optymalnych, podczas gdy w głowie funkcji znalazł się pierwszy i kolejne fronty Pareto-optymalne.

Warto również zwrócić uwagę na fakt, iż bardzo mała liczba ze znalezionych przez algorytmy rozwiązań, była optymalna w sensie Pareto.

Literatura

1. Deb K., Pratap A., Agarwal S., Meyarivan T.: A fast and elitist multi-objective genetic algorithm NSGA-II. IEEE Transaction on Evolutionary Computation, 6(2), 2002, 181-197.
2. Garey M.R., Johnson D.S.: A Guide to the Theory of NP-Completeness, CA: Freeman, San Francisco, 1979.
3. Knowles, J. Corne, D., The Paretoarchi ved evolution strategy: A new baseline algorithm for multiobjective optimisation. Congress on Evolutionary Computation, Piscatway: New Jersey: IEEE Service Center, 1999 , 98–105.
4. Lageweg B. J., Lenstra J. K., Rinnooy Kan A. H. G., A general bounding scheme for the permutation flow-shop problem, Oper. Res., vol. 26, no. 1, 1978, 53–67.
5. Pinedo M.: Scheduling: Theory, Algorithms and Systems, 2end ed., NJ: Prentice-Hall, Englewood Cliffs, 2002.
6. Rudolph G.: Evolutionary search under partially ordered sets. Technical Report No. CI-67/99, Dortmund:Department of Computer Science/LS11, University of Dortmund, Germany, 1999.
7. Schaffer J.D. Multiple Objective Optimization with Vector Evaluated Genetic Algorithms . - New Jersey : L. Erlbaum Associates Inc. Hillsdale, 1985.
8. Taillard E.: Benchmarks for basic scheduling problems. European Journal of Operational Research vol. 64, 1993, 278-285.
9. Zitzler E., Deb K., Thiele L.: (in press) Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary Computation, 8.
10. Zitzler E. Thiele L.: Multiobjective optimization using evolutionary Algorithms - A comparati ve case study. In Eiben, A. E., Back, T., Schoenauer , M., and Schwefel, H.-P., editors, Parallel Problem Solving from Nature, V, 292–301, Springer , Berlin, Germany, 1998.

Mgr inż. Dominik ŻELAZNY
Instytut Informatyki, Automatyki i Robotyki
Politechnika Wrocławska
50-370 Wrocław, ul. Janiszewskiego 11/17
Tel: (0-71) 320 27 45 / 321 26 77
e-mail: dominik.zelazny@pwr.wroc.pl