AUTOMATYKA • 2011 • Tom 15 • Zeszyt 2

Wojciech Chmiel*, Piotr Kadłuczka*, Grzegorz Packanik*

Zastosowanie algorytmów rojowych w rozwiązywaniu zagadnień permutacyjnych

1. Wprowadzenie

Przedstawione w artykule algorytmy populacyjne stanowią alternatywę dla klasycznych algorytmów ewolucyjnych. Celem przeprowadzonych eksperymentów obliczeniowych było porównanie efektywności algorytmów rojowych (ptasiego i pszczelego) oraz ich dostosowanie do specyfiki problemów permutacyjnych. Zagadnienia permutacyjne tj. komiwojażera, przydziału (także QAP) oraz szeregowania zadań, stanowiące ważną grupę problemów optymalizacji dyskretnej, charakteryzują się jednak zróżnicowanymi własnościami, mającymi wpływ na konstrukcje efektywnych algorytmów.

1.1. Model zagadnienia permutacyjnego

Jako przykład testowy zagadnienia permutacyjnego przyjęto zagadnienie szeregowanie zadań. Modeluje ono bardzo zróżnicowane problemy rzeczywiste: począwszy od elastycznych systemów produkcyjnych, planowania produkcji, projektowania systemów komputerowych, logistyki czy komunikacji. Skupiając się na problemie szeregowania zadań na maszynach, można go opisać w następujący sposób [1]:

Należy zaplanować wykonanie n zadań (składających się z m operacji) na m maszynach tak, aby zminimalizować zadane kryterium jakości, przy spełnieniu następujących założeń:

- każde zadanie musi być przetwarzane na każdej maszynie jeden i tylko jeden raz,
- kolejność wykonywania operacji jest identyczna dla każdego zadania oraz każdej maszyny,
- nie można rozpocząć następnej operacji wchodzącej w skład zadania k, dopóki poprzednia operacja nie zostanie zakończona,
- zadania aktualnie wykonywane na maszynie nie mogą być wywłaszczane w celu wykonania operacji wchodzącej w skład innego zadania,
- znane są czasy wykonania wszystkich zadań na wszystkich maszynach.

* AGH Akademia Górniczo-Hutnicza, Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki, Katedra Automatyki

Dodatkowym założeniem jest to, iż czasy przezbrojeń nie zależą od kolejności, w jakiej przetwarzane jest zadanie, co niestety w rzeczywistych systemach zdarza się niezwykle rzadko. Tego typu problemy szeregowania zadań określane są mianem *system-dependent set-up times* (SDST), przykładem takiego problemu jest montaż elementów na płytce PCB. Dla uproszczenia rozpatrywanego modelu założono, że czas przezbrojenia jest zawarty w czasie przetwarzania i nie zależy od pozycji, na której zadanie zostało uszeregowane.

Kolejnym wariantem tego problemu jest założenie o braku magazynów między stanowiskami. Oznacza to, że zadanie *i,* którego przetwarzanie właśnie kończy się na maszynie *r*, nie może przejść na maszynę $r+1$, jeśli na tej maszynie nie skończyło się przetwarzanie poprzednika zadania *i*. Zadanie *i* pozostaje zatem na maszynie *r*, wstrzymując przetwarzanie kolejnego zadania na tej maszynie, do czasu kiedy zwolniona zostanie maszyna $r+1$. W rozpatrywanym modelu założono istnienie zapasów między stanowiskowych o nieskończonej pojemności. Dzięki temu, gdy przetwarzanie zadania *i* skończy się na maszynie *r*, a rozpoczęcie jego przetwarzania będzie chwilowo niemożliwe na maszynie $r+1$, zostanie ono przeniesione na zapas międzystanowiskowy i tam będzie oczekiwać na możliwość przetwarzania, pozwalając jednocześnie na przetwarzanie kolejnego zadania na maszynie *r*.

Rozwiązaniem problemu jest permutacja zadań (ich kolejność) na pierwszej maszynie, ponieważ kolejność przetwarzania na kolejnych maszynach jest z góry ustalona. Poszukiwana jest taka permutacja, która minimalizuje zadany wskaźnik jakości. Stosowane kryteria oceny jakości uszeregowania to [2] – długość uszeregowania (maksymalny czas zakończenia – przepływu):

$$\Phi(\pi) = \max_{1 \le i \le n} \{ f_i(\pi) \} \tag{1}$$

lub średni czas przepływu (tzn. średni czas przebywania zadania w systemie)

$$\overline{\Phi(\pi)} = \frac{1}{n} \sum_{i=1}^{n} f_i(\pi) \tag{2}$$

gdzie: $f_i(\pi)$ – czas zakończenia *i*-tego zadania dla uszeregowania π.

Rozpatrywanym kryterium jakości jest (1) maksymalny czas zakończenia przetwarzania zadań, w literaturze często oznaczany jako $c_{\max}$. Problem ten należy do klasy zagadnień *NP*-trudnych i do jego rozwiązywania stosuje się algorytmy przybliżone.

2. Algorytmy rojowe

Algorytmy należące do tej grupy naśladują zachowania populacyjne zwierząt, np. sposobu poszukiwania pożywienia. Okazuje się, że bardzo proste i słabe indywidualnie osobniki, dzięki współpracy i przekazywaniu informacji między sobą, mogą poradzić sobie z rozwiązaniem złożonych problemów. W trakcie tego procesu poszczególne osobniki

w celu wyznaczenia kierunku poszukiwania pożywienia biorą pod uwagę dwa czynniki, własne doświadczenie oraz najlepsze miejsce znalezione przez całą populację. Wśród algorytmów należących do tej grupy można wyróżnić:

- Algorytm pszczeli – symulujący proces poszukiwania pożywienia przez roje pszczół miodnych. W algorytmie tym poszczególne pszczoły przekazują sobie informacje o miejscach obfitych w pożywienie za pomocą tańca pszczół. W najbardziej obiecujące miejsca są następnie wysyłane kolejne osobniki.
- Algorytm mrówkowy – symuluje zachowanie koloni mrówek, które za pomocą feromonów oznaczają najkrótszą drogę prowadzącą do pożywienia.
- Algorytm ptasi – naśladujący stadne zachowania ptaków, które komunikując się oraz wzajemnie obserwując, wymieniają pomiędzy sobą informacje, usprawniając przeszukiwanie teren (przestrzeni rozwiązań).

2.1. Algorytm pszczeli

Algorytm *BA* (*bees algorithm*) został zainspirowany poprzez proces poszukiwania pożywienia, jaki stosują roje pszczół miodnych. Kolonia pszczół przeszukuje znaczną przestrzeń, sprawdzając jednocześnie lokalizacje w różnych kierunkach odległe nawet do 10 kilometrów. Lokalizacje, gdzie jest dużo pożywienia lub te bliższe, są odwiedzane przez większą liczbę robotnic. W pierwszej kolejności w otoczenie ula są wysyłane pszczoły – zwiadowcy, których zadaniem jest losowe przeszukiwanie terenu oraz przekazywanie informacji kolonii o znalezionych źródłach pożywienia za pomocą tańca pszczół. W trakcie tańca pszczoły przekazują trzy istotne informacje określające znalezioną przez siebie lokalizację: kierunek w jakim znajduje się pożywienie, odległość oraz jakość źródła pokarmu. Po wykonaniu swojego tańca zwiadowca dołącza do pszczół miodnych czekających wewnątrz ula. Kolonia następnie na podstawie uzyskanych informacji decyduje, ile robotnic należy wysłać do konkretnych lokalizacji. Liczba ta jest określana na podstawie dwóch kryteriów: jakości danej lokalizacji oraz energii jaką trzeba zużyć, aby zebrać z niej pożywienie. Więcej robotnic jest wysyłanych do obiecujących lokalizacji, co pozwala na szybsze i bardziej efektywne zbieranie pożywienia. Podczas zbierania pożywienia robotnice cały czas monitorują stan źródła pokarmu, przekazując na bieżąco informacje pozostałym osobnikom, gdy tylko wrócą do ula. Pozwala to na podejmowanie kolejnych decyzji co do liczby wysyłanych pszczół lub jeśli pokarm w danej lokalizacji wyczerpie się, o wysłaniu zwiadowców do poszukiwania nowych miejsc obfitych w pożywienie [3].

Schemat podstawowy algorytmu pszczelego pozwala na znaczną swobodę jego implementacji w zakresie [4]:

- sposobu utworzenia populacji początkowej,
- zastosowanego mechanizmu selekcji,
- definicji przeszukiwanego sąsiedztwa,
- sposobu utworzenia nowej populacji,
- warunku zakończenia obliczeń.

Szczegóły dotyczące implementacji powyższych elementów oraz dobór parametrów sterujących pracą algorytmu decyduje o jego efektywności. W opisywanej wersji algorytmu rozwiązania początkowe są tworzone w sposób losowy i zapisane w plikach tekstowych. Stanowią one populacje początkowe dla porównywanych algorytmów o różnych parametrach.

Kluczowym elementem decydującym o efektywności algorytmu jest dobór przeglądanego sąsiedztwa oraz jego wielkość. Liczby nowych rozwiązań, jakie zostaną sprawdzone dla danych lokalizacji określana jest na podstawie wartości funkcji celu. W posortowanej populacji wybieranych jest l_e najlepszych rozwiązań, nazywanych dalej lokalizacjami elitarnymi oraz l_b lokalizacji dobrych. Wielkość sąsiedztwa dla rozwiązań elitarnych wynosi s_e , natomiast dla rozwiązań dobrych s_b. Zarówno l_e, l_b, jak i s_e, s_b są parametrami algorytmu. Pozostałe lokalizacje uznawane są za mało obiecujące i nie są do nich wysyłane dodatkowe osobniki. W algorytmie (rys. 1) stosuje się operatory genetyczne (unarne) specjalizowane dla problemów permutacyjnych o następującym sposobie generacji sąsiedztwa:

- przestawienie losowo wybranego zadania na losowo wybraną pozycję, pozostałe zadania są przesuwane o jedną pozycję w stronę przeciwną niż nastąpiło przesunięcie wylosowanego zadania;
- przestawienie dwóch losowo wybranych elementów rozwiązania;
- cykliczne przesunięcie rozwiązania (rotacja) w zadanym kierunku, o zadaną liczbę pozycji;
- generacja następnika permutacji w sensie leksykalnym, w tym przypadku jeśli wygenerowanych ma być więcej niż jedno rozwiązań należące do sąsiedztwa, to operator ten jest stosowany na poprzednio wygenerowanym następniku, tak aby nie uzyskać powtórzeń rozwiązań;
- generacja poprzednika permutacji w sensie leksykalnym, także w tym przypadku generacja kolejnych rozwiązań odbywa się analogicznie jak w punkcie powyższym;
- przestawienie bloków o losowej długości, w oparciu o wylosowane dwie pozycje określające początki bloków do wymiany, kolejność zadań w blokach może być dodatkowo odwracana;
- odwrócenie kolejności elementów w bloku o losowej długości, znajdującego się na losowej pozycji.

Procentowy udział w sąsiedztwie rozwiązań generowanych za pomocą każdego operatora jest parametrem algorytmu. Kolejna populacja osobników jest tworzona poprzez wybór najlepszych osobników ze wszystkich lokalizacji elitarnych i dobrych, a brakująca część jest generowana w sposób losowy.

Zaimplementowany został również dodatkowy mechanizm zabezpieczający przed utykaniem algorytmu w minimach lokalnych. Każdy osobnik może przeszukiwać przestrzeń tylko przez określoną przez użytkownika liczbę cykli – długość życia, po której zostaje on zastąpiony nowym, losowo wygenerowanym rozwiązaniem. Zapamiętywane jest historycznie najlepsze rozwiązanie.

Zastosowanym kryterium zakończenia pracy algorytmu jest przekroczenie dopuszczalnej liczby iteracji (nowych rozwiązań).

1. Generacja populacji początkowej λ rozwiązań:
   - losowe utworzenie λ permutacji,
   - ocena rozwiązań należących do populacji początkowej,
   - posortowanie rozwiązań według wartości funkcji celu,
   - zapisanie aktualnie najlepszego rozwiązania do π_{best}.
2. Dla $l_e + l_b$ najlepszych rozwiązań:
   - definicja sąsiedztwa aktualnie przetwarzanego rozwiązania π: $S(\pi)$,
   - wybór najlepszego rozwiązania należącego do sąsiedztwa $S(\pi)$: $\pi'' = \arg\max\{\phi(\pi'): \pi' \in S(\pi)\}$
3. Generacja nowej populacji:
   - wybór $l_e + l_b$ najlepszych rozwiązań (jedno z każdej badanej lokalizacji),
   - losowe utworzenie $\lambda - (l_e + l_b)$ brakujących rozwiązań,
   - sprawdzenie czy któryś z osobników nie przekroczył dopuszczalnego czasu życia LT, jeśli tak to zastąpienie go nowym, losowym rozwiązaniem,
   - posortowanie nowej populacji według wartości funkcji celu.
4. Jeśli nastąpiła poprawa wartości funkcji celu, uaktualnienie π_{best}.
5. Sprawdzenie warunku zakończenia obliczeń (czy nie została przekroczona dopuszczalna ilość iteracji I_{max})
   - jeśli tak to koniec działania algorytmu i zwrócenie π_{best} oraz $\phi(\pi_{best})$,
   - jeśli nie to powrót do punktu 2.

Rys. 1. Schemat algorytmu pszczelego – *BA*

2.2. Algorytm ptasi

Algorytm *PSO* (*Particle Swarm Optimalization*) został po raz pierwszy przedstawiony w 1995 roku (Kennedy, Eberhart – [5]) i od tego czasu cieszy się niesłabnącym zainteresowaniem naukowców.

Inspiracją algorytmu jest zachowanie stad ptaków lub ławic ryb. Naukowcy zauważyli, że osobniki w stadzie mają tendencję do utrzymywania optymalnych odległości od swoich sąsiadów, dzięki odpowiedniemu dostosowaniu swojej prędkości. Podstawowym elementem *PSO* jest stado osobników (zbiór rozwiązań) „lecących" przez wielowymiarową przestrzeń z prędkością, która określa, gdzie dany osobnik znajdzie się w następnej iteracji. Pozycja *i*-tego osobników w iteracji *t* wyznaczana jest za pomocą wzoru [6]:

$$x_i^t = x_i^{t-1} + v_i^t \tag{3}$$

gdzie prędkość osobnika wyrażona jest wzorem:

$$v_i^t = \omega v_i^{t-1} + c_1 r_1 (p_i^{t-1} - x_i^{t-1}) + c_2 r_2 (p_g^{t-1} - x_i^{t-1}) \tag{4}$$

gdzie:

ω – współczynnik bezwładności,

c_1, c_2 – wagi świadomości oraz myślenia społecznego,

r_1, r_2 – liczby losowe z przedziału [0, 1],

p_i^{t-1} – najlepsze dotychczas znalezione przez danego osobnika rozwiązanie,

p_g^{t-1} – najlepsze rozwiązanie znalezione przez całe stado.

PSO świetnie nadaje się do rozwiązywania problemów ciągłych, w przypadku problemów dyskretnych pewne elementy muszą zostać zmodyfikowane lub przedefiniowane.

W opisywanym algorytmie redefinicji uległ sam osobnik, a dokładniej sposób określania jego pozycji oraz definicja prędkości, transformująca jedną permutację zadań w inną. Każdy z osobników składa się z następujących elementów:

- aktualnego położenia (rozwiązania),
- aktualnej prędkości (ograniczoną prędkością maksymalną),
- najlepszej znalezionej przez siebie lokalizacji (rozwiązania),
- oceny aktualnej lokalizacji (wartości funkcji celu).

W zaimplementowanym algorytmie postać permutacyjną rozwiązania zastąpiono postacią macierzową, gdzie w macierzy $\boldsymbol{X_i}_{[nxn]}$ aktualne położenie *i*-tego osobnika (uszeregowanie zadań) zapisujemy następująco:

$x_{i,j,k} = 1$, gdy zadanie *j*-te jest uszeregowane na pozycji *k*, w przeciwnym przypadku $x_{i,j,k} = 0$.

Prędkość osobnika *i* zdefiniowana jako:

$$v_i^t = (v_{i,1,1}^t, v_{i,1,2}^t, ..., v_{i,n,n}^t);\ v_{i,j,k}^t \in \mathbb{R} \tag{5}$$

gdzie $v_{i,j,k}^t$ jest wartością prędkości osobnika *i* w iteracji *t* dla zadania *j* umieszczonego na pozycji *k*.

Definicja prędkości została zainspirowana mechanizmem pamięci częstotliwościowej algorytmu *Taboo Search*, w którym zliczamy, ile razy dane zadanie zostało uszeregowane na danej pozycji. Duża wartość elementu $v_{i,j,k}^t$ zwiększa prawdopodobieństwo uszeregowania zadania *j* na pozycji *k*. Wartość prędkości w kolejnych iteracjach wyznaczana jest ze wzoru [7]:

$$v_{i,j,k}^t = \omega v_{i,j,k}^{t-1} + c_1 r_1 (p_{i,j,k}^t - x_{i,j,k}^t) + c_2 r_2 (p_{g,j,k}^t - x_{i,j,k}^t) \tag{6}$$

gdzie:

$p_i^t = (p_{i,1,1}^t, p_{i,1,2}^t, ..., p_{i,n,n}^t)$; $p_{i,j,k}^t \in \{0,1\}$ – macierz permutacyjna określająca najlesze dotychczas znalezione rozwiązanie przez danego osobnika,

$p_g^t = (p_{g,1,1}^t, p_{g,1,2}^t, ..., p_{g,n,n}^t)$; $p_{g,j,k}^t \in \{0,1\}$ – macierz permutacyjna dla najlepszego dotychczas znalezionego przez całe stado rozwiązania.

Wzór (6) pozwala uwzględnić w prędkości zarówno własne doświadczenie, jak i świadomość społeczną stada. Taka postać prędkości nie nadaje się jeszcze do konstrukcji nowego rozwiązania, musi być ona jeszcze konwertowana do macierzy S_i^t określającej prawdopodobieństwo uszeregowania zadań na danych pozycjach. Uzyskuje się to przez podstawienie elementów macierzy V_i^t jako argumentów sigmoidy określonej następującym wzorem:

$$s(v_{i,j,k}^t) = \frac{1}{1+e^{-v_{i,j,k}^t}} \tag{7}$$

Wartość $s(v_{i,j,k}^t)$ określa prawdopodobieństwo przyjęcia przez element $x_{i,j,k}^t$ wartości 1. Konstrukcja nowego rozwiązania, w klasycznym podejściu, rozpoczyna się od pustej sekwencji, ustawiając kolejne zadania na pozycjach zgodnie z prawdopodobieństwem określonym następująco:

$$q_i^t(j,k) = \frac{s(v_{i,j,k}^t)}{\sum_{j \in U} s(v_{i,j,k}^t)} \tag{8}$$

gdzie U jest zbiorem wszystkich zadań, które nie zostały jeszcze uszeregowane. Operacja ta jest powtarzana do momentu zbudowania pełnej sekwencji zadań.

W implementowanym algorytmie, w celu ograniczenia nakładu obliczeniowego, sposób budowania sekwencji uległ niewielkiej modyfikacji. We wzorze (8) zbiór U został zastąpiony mniej licznym zbiorem F, zawierającym jedynie f pierwszych nieuszeregowanych zadań w rozwiązaniu p_g^t.

Parametrami algorytmu ptasiego przedstawionego na rysunku 2 są:

λ – wielkość populacji,

$I_{\max}$ – zadana liczba iteracji,

$C_0, C_1, C_2, V_{\max}$ – parametry wyznaczania prędkości,

f – długość listy zadań rozważanych do uszeregowania na danej pozycji.

1. Generacja populacji początkowej λ rozwiązań:
   - losowe utworzenie λ permutacji oraz zainicjowanie prędkości,
   - ocena rozwiązań należących do populacji początkowej,
   - posortowanie rozwiązań według wartości funkcji celu,
   - zapisanie aktualnie najlepszego rozwiązania do P_g^0.
2. Uaktualnienie prędkości oraz pozycji:
   - obliczenie aktualnej prędkości,
   - obliczenie prawdopodobieństw zmian,
   - budowa nowego rozwiązania.
3. Ocena rozwiązań oraz aktualizacja P_i oraz P_g:
   - ocena rozwiązań
   - jeśli $\Phi(X_i) < \phi(P_i)$, podstaw $P_i = X_i$
   - jeśli $\Phi(P_g) > argmin\{\Phi(\mathrm{X}_1), \Phi(\mathrm{X}_2), \dots, \Phi(\mathrm{X}_\lambda)\}$, dokonaj podstawienia $P_g = argmin\{\Phi(\mathrm{X}_1), \Phi(\mathrm{X}_2), \dots, \Phi(\mathrm{X}_\lambda)\}$.
4. Sprawdź warunek stopu.
   - jeśli osiągnięta została maksymalna dopuszczalna liczba iteracji to koniec, zwróć π_{best} oraz $\phi(\pi_{best})$
   - jeśli nie to powrót do punktu 2.

Rys. 2. Schemat algorytmu ptasiego – *PSO*

3. Wyniki eksperymentów obliczeniowych

Eksperymenty obliczeniowe [8] zostały przeprowadzone w oparciu o instancje testowe dla zagadnienia szeregowania zadań opracowane przez E. Taillarda [9]. Dla każdej instancji problemu znane są rozwiązania referencyjne, pozwalające sprawdzić jakość uzyskanych wyników dla zaimplementowanych algorytmów. W trakcie wstępnie przeprowadzonych testów dobrano parametry:

dla algorytmu *BA*:

$\lambda = 100$, $I_{\max} = 100$, $l_e = 3$, $s_e = 50$, $l_b = 10$, $s_b = 10$, pełny zbiór operatorów;

dla algorytmu *PSO*:

$\lambda = 200$, $I_{\max} = 160$, $C_0 = 0.8$, $C_1 = 0.5$, $C_2 = 0.5$, $V_{\max} = 4$, $f = 0.25\ n$.

Następnie porównano działanie obu algorytmów, prezentując w tabeli 1 uzyskane wyniki dla wybranych reprezentatywnych 10 zadań testowych: *T01*, *T02*, *T05*, *T17*, *T19*, *T31*, *T35*, *T62*, *T64*, *T91*. Przyjęte oznaczenia:

Φ_{ref} – wartość funkcji celu podana dla instancji testowej jako wartość odniesienia.

Φ_{best} – najlepsza wartość funkcji celu jaka została uzyskana w 5 przebiegach algorytmu

$\Phi_{śr}$ – średnia wartość funkcji celu najlepszego rozwiązania dla 5 przebiegów algorytmu

$$E[\%] = \frac{\Phi_{best} - \Phi_{ref}}{\Phi_{ref}},$$

$$E_{śr}[\%] = \sum_{i \in S} \frac{E_i}{|S|}; \quad \text{gdzie } S = \{T_{01}, T_{02} ..., T_{91}\}.$$

Analizując zebrane w tabeli 1 wyniki, można zauważyć, iż dla każdej instancji testowej lepsze rezultaty uzyskał algorytm ptasi. Dodatkowym jego atutem jest krótszy czas obliczeń. Do jego wad można zaliczyć znaczne zapotrzebowanie na pamięć szczególnie w przypadku dużych problemów. Zaletą algorytmu pszczelego jest to, że wynik jego działania jest w mniejszym stopniu zależny od równomiernego rozłożenia populacji początkowej w przestrzeni rozwiązań, podczas gdy algorytm ptasi wymaga większego zróżnicowania populacji początkowej.

Tabela 1
Porównanie działania algorytmów *BA* oraz *PSO* dla wybranych zadań testowych

		Algorytm pszczeli				Algorytm ptasi			
Nazwa	Φ_{ref}	Φ_{best}	$E[\%]$	$\Phi_{śr}$	$E[\%]$	Φ_{best}	$E[\%]$	$\Phi_{śr}$	$E[\%]$
T01	1278	1279	0,1	1297	1,5	1289	0,9	1292	1,1
T02	1359	1373	1,0	1375	1,2	1359	0,0	1365	0,4
T05	1236	1267	2,5	1279	3,5	1244	0,6	1249	1,0
T17	1484	1577	6,3	1596	7,6	1498	0,9	1522	2,6
T19	1593	1676	5,2	1691	6,1	1634	2,6	1646	3,3
T31	2724	2774	1,8	2799	2,8	2730	0,2	2740	0,6
T35	2863	2928	2,3	2939	2,7	2887	0,8	2889	0,9
T62	5268	5462	3,7	5483	4,1	5327	1,1	5366	1,9
T64	5014	5160	2,9	5216	4,0	5054	0,8	5067	1,1
T91	10868	11546	6,2	11602	6,8	11297	3,9	11325	4,2
$E_{śr}[\%]$			3,2		4,0		1,2		1,7

4. Podsumowanie

Implementacja oraz porównanie wyników działania dwóch algorytmów należących do grupy algorytmów rojowych dla *NP*-trudnego zagadnienia szeregowania zadań potwierdza ich efektywność.

Algorytm pszczeli jest dość prosty w implementacji dla rozwiązywania problemów permutacyjnych – można w nim użyć specjalizowane operatory genetyczne. Uzyskane wyniki (średni błąd na poziomie 4%) są zadowalające, jeśli konieczne jest uzyskanie wyników w krótkim czasie.

Implementacji algorytmu ptasiego, ze względu na jego ukierunkowanie w stronę optymalizacji zagadnień o naturze ciągłej, wymagało przedefiniowania podstawowych elementów algorytmu i dostosowanie ich do specyfiki problemów permutacyjnych. Uzyskiwane przez algorytm wyniki były bardzo dobre: średni błąd na poziomie 1% wartości referencyjnej oraz stosunkowo krótki czas obliczeń są akceptowalne w większości zastosowań praktycznych.

Literatura

[1] Filipowicz B., *Badania operacyjne. Wybrane metody obliczeniowe i algorytmy. Część I*. Kraków, F.U.H Poldex, 1997.

[2] Coffman jra E.G., *Teoria szeregowania zadań.* Warszawa, Wydawnictwa Naukowo-Techniczne, 1980.

[3] Pham D.T., Koç E., Lee J.Y., Phrueksanant J., *Using the Bees Algorithm to schedule jobs for a machine*. Manufacturing Engineering Centre, Cardiff University, 2007.

[4] Filipowicz B., Chmiel W., Kadłuczka P., *Ukierunkowane przeszukiwanie przestrzeni rozwiązań w algorytmach rojowych*. Automatyka (półrocznik AGH), 13, 2, 2009.

[5] Kennedy j., Eberhart R.C., *Particle Swaem Optimization*. In Proceedings of the IEEE International Conference on Neural Networks, Perth, 1995, 12.

[6] Ismael A., Vaz F., Vicente L.N., *A particle swarm pattern search method for bound constrained nonlinear optimization.* Portugalia, 2006.

[7] Ching-Jong L., Chao-Tang T., Pin L., *A discrete version of particle swarm optimization for flowshop scheduling problems*. Computers & Operations Research, 2007.

[8] Packanik G., *Zastosowania algorytmów rojowych w zagadnieniach szeregowania zadań.* Praca magisterska, AGH, Kraków, 2010.

[9] Tillard E., *Benchmarks for Basic Scheduling Problems*. http://mistic.heig-vd.ch/taillard/problemes.dir/ ordonnancement.dir/ordonnancement, 1989.