I Krajowa Konferencja
Algorytmy Ewolucyjne
Murzasichle, 12-15.06.1996

Wykorzystanie algorytmu genetycznego do poprawy rozwiązania zadania minimalizacji długości uszeregowania zadań niezależnych i niepodzielnych.

Jarosław Stańczak
Instytut Podstaw Elektroniki
Politechnika Warszawska
ul. Nowowiejska 15/19
00-665 Warszawa
stanczak@ipe.pw.edu.pl

Wojciech Wikarczyk
Instytut Podstaw Elektroniki
Politechnika Warszawska
ul. Nowowiejska 15/19
00-665 Warszawa
wwik@ipe.pw.edu.pl

1. Streszczenie

Zadanie minimalizacji długości uszeregowania należy do problemów NP-zupełnych. Oznacza to, że nie istnieją efektywne[1] algorytmy optymalnego rozwiązania tego problemu. Istnieją natomiast dość szybkie i efektywne algorytmy dające rozwiązania suboptymalne. Jednakże rozwiązania te mogą w pewnych warunkach dość znacznie odbiegać od rozwiązań optymalnych. Otrzymane za ich pomocą wyniki można jednak poprawić wykorzystując w tym celu algorytmy genetyczne. W niniejszej pracy zbadano możliwości zastosowania algorytmów genetycznych do rozwiązywania problemu szeregowania zadań niezależnych i niepodzielnych o znanych a priori czasach wykonywania w systemie wieloprocesorowym w celu minimalizacji długości uszeregowania.

2. Zagadnienie szeregowania zadań

Problem uszeregowania zadań w procesie produkcyjnym lub systemie komputerowym, minimalizującego jakieś kryterium jakości jest bardzo częstym zagadnieniem spotykanym w zastosowaniach technicznych. Istnieje znaczne zróżnicowanie tego typu zagadnień w zależności od stosowanego kryterium jakości, ograniczeń i zależności między zadaniami lub od samego charakteru problemu - deterministycznego, czy też probabilistycznego.

Rozpatrywane w tej pracy problemy należą do klasy zagadnień szeregowania deterministycznego. Oznacza to, że znane są parametry zadań oczekujących na obsługę.

W omawianym zagadnieniu rozpatrywany jest zbiór n zadań $Z = \{z_1, z_2, \ldots, z_n\}$, wykonywanych na m procesorach, tworzących zbiór $P = \{p_1, p_2, \ldots, p_m\}$.

---

[1] Algorytm efektywny, to algorytm o złożoności wielomianowej, czyli taki w którym nakład obliczeń potrzebny do rozwiązania problemu zależy w sposób wielomianowy od wielkości zadania.

Każde zadanie scharakteryzowane jest przez wektor $T_j = \{t_{1j}, t_{2j}, .., t_{ij}, .., t_{mj}\}$, określający czas wykonywania zadania z_j na procesorze p_i. Dla każdego zadania określony jest czas zakończenia wykonywania d_j, czasy przybycia zadań do systemu są jednakowe. Nie ma ograniczeń kolejnościowych, wymuszających określone sekwencje w wykonywaniu zadań. Zadania są niepodzielne, czyli nie jest dopuszczane ich przerywanie.

Procesory należą do klasy procesorów jednorodnych, co oznacza, że każdy z nich jest w pełni uniwersalny, różnice między nimi dotyczą jedynie prędkości, z jaką pracują. Prędkość ta jest niezależna od rodzaju wykonywanego zadania i może być przedstawiona w postaci wektora, pokazującego wartość stosunku prędkości danego procesora do procesora najwolniejszego $B = \{b_1, b_2, ..., b_m\}$.

Rozwiązaniem problemu jest znalezienie uszeregowania zadań, czyli przyporządkowanie zadań procesorom, które będą je wykonywały. Rozwiązanie dopuszczalne musi spełniać pewne założenia. W omawianym przypadku istotna jest niepodzielność zadań, konieczność wykonania wszystkich zadań z kolejki oraz intuicyjny warunek dopuszczający wykonywanie tylko jednego zadania w danym momencie na jednym procesorze.

Niech moment zakończenia pracy przez procesor j zostanie oznaczony symbolem c_j. Powstanie w ten sposób wektor czasów zakończenia pracy przez wszystkie procesory $C = \{c_1, c_2, ..., c_m\}$. Można także zdefiniować parametr, będący oszacowaniem od dołu parametrów uszeregowania. Powstaje on przez odrzucenie warunku niepodzielności zadań:

$$C^* = \frac{\sum_{j=1}^{m} c_j}{m}$$

Kryterium jakości rozpatrywanego problemu jest minimalizacja długości uszeregowania zadań, czyli:

$$C_m = \min_j \left\{ \max_j (c_j) \right\}$$

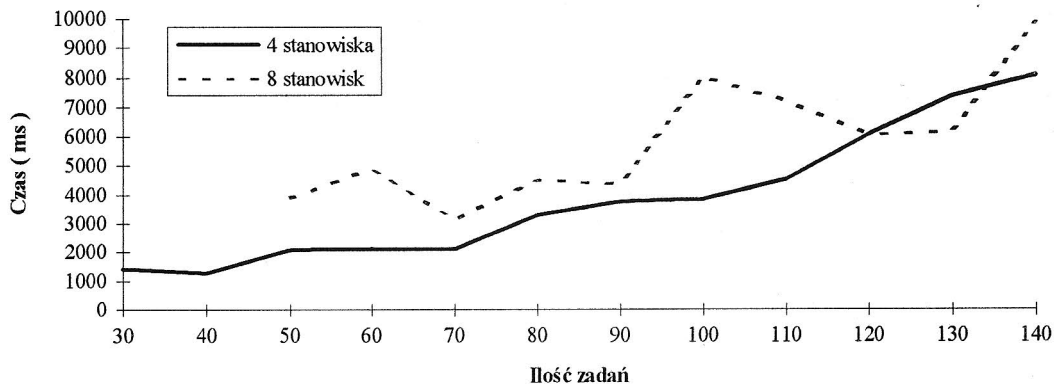
3. Klasyczne algorytmy genetyczne w szeregowaniu zadań

W celu zbadania skuteczności działania AG w zastosowaniach szeregowania zadań w wielostanowiskowych systemach masowej obsługi przeprowadzono następujące eksperymenty obliczeniowe :

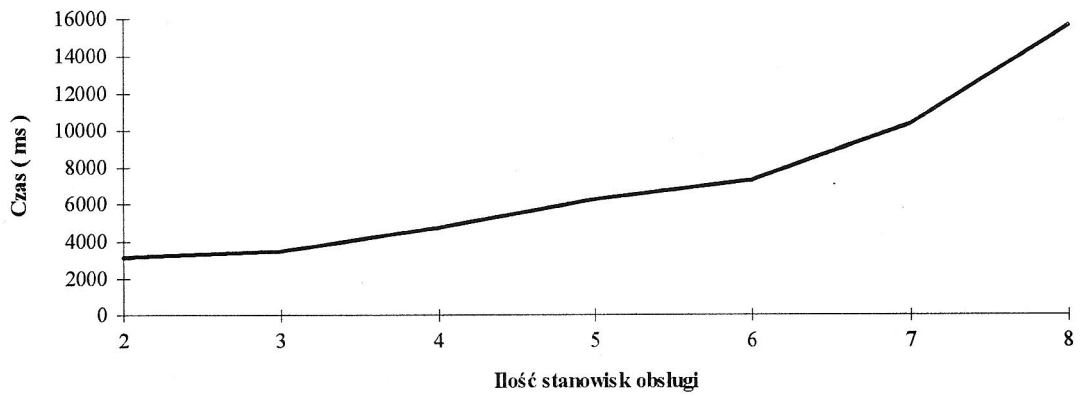
a) zaimplementowano algorytm genetyczny minimalizujący długość uszeregowania zadań w m stanowiskowym systemie obsługi.

Przyjęto kombinowany wariant kodowania. Zadania zakodowano w postaci tablicy rekordów: jedno jego pole zawierało identyfikator zadania, drugie pole zawierało numer stanowiska obsługi na którym zadanie należało wykonać. Aby uniemożliwić przypadkowe powstawanie zadań o tym samym identyfikatorze zastosowano operator krzyżowania PXM [2], jako operator mutacji zastosowano operację zamieniającą miejscami dwa wylosowane rekordy. Uniemożliwiało to również powstanie dwóch identycznych rekordów. Operator ten nazwano MUTE1. Wybór numeru stanowiska na którym zadanie wykonywano był zmieniany za pomocą operatora mutacji. Ponieważ nie istnieją ograniczenia ilości wykonywanych zadań na jednym stanowisku nie zaistniały żadne ograniczenia związany z charakterem tego operatora.

Użyto operator mutacji losowej. Przyjęto binarną reprezentację numeru stanowiska. Operator ten nazwano MUTE2.



Wykres 1. Zależność pomiędzy czasem osiągnięcia rozwiązania a ilością zadań dla 4 i 8 stanowiskowych systemów obsługi (PENTIUM 90).



Wykres 2. Zależność pomiędzy czasem osiągnięcia rozwiązania a ilością stanowisk obsługi (PENTIUM 90).

b) Zbadano szybkość działania AG w zależności od ilości stanowisk m i ilości zadań n. Czas wykonywania zadań generowano losowo z rozkładem jednostajnym z przedziału ⟨5,20⟩. Losowo generowany był również czas przybycia zadania do systemu. Wahał się on w przedziale ⟨0,10⟩. Przeprowadzono obliczenia dla ilości stanowisk w przedziale ⟨2,8⟩. Jako ilość zadań przyjęto 50 a jako rozmiar populacji 35. Obliczenia były prowadzone do momentu uzyskania wyniku lepszego od rozwiązania suboptymalnego otrzymywanego za pomocą algorytmu LPT (*longest procesing time*) [1].

3.1. Wnioski

Na podstawie przeprowadzonych doświadczeń można stwierdzić iż algorytmy genetyczne są zbyt wolne w przypadku zastosowania ich do szeregowania zadań np. w systemach operacyjnych. Nakład obliczeń potrzebny na wygenerowanie dość dobrego rozwiązania jest

niewspółmiernie większy niż w przypadku zastosowania algorytmów przybliżonych lecz działających ze złożonością wielomianową.

4. Algorytmy genetyczne w problemach optymalizacyjnych

Zagadnienie minimalizacji czasu wykonania zbioru zadań niepodzielnych w systemie wieloprocesorowym jest problemem NP-zupełnym. Oznacza to, że wykorzystywane w praktyce algorytmy, rozwiązujące tego typu problemy, dają jedynie rezultat przybliżony, często dość odległy od optymalnego. Nie dają one żadnej gwarancji znalezienia rozwiązania optymalnego, choć w wielu przypadkach są dość „tanie" obliczeniowo. Przykładem takich algorytmów może być algorytm LPT dla przypadku procesorów jednakowych i jego uogólnienie na przypadek procesorów jednorodnych, podane np. w pracy [1].

Algorytmy genetyczne są dość często wykorzystywane w procesach optymalizacji. Są one oparte na mechanizmach doboru naturalnego i dziedziczenia cech, czyli głównych cechach warunkujących ewolucję organizmów żywych. Podobnie jak to ma miejsce w przyrodzie, wykorzystując wymianę materiału genetycznego między osobnikami w trakcie rozmnażania oraz przypadkowe mutacje powstają osobniki o nowych właściwościach, których przydatność sprawdza środowisko. W przypadku algorytmu genetycznego sprawdzianem tym jest funkcja dopasowania, oparta na funkcji celu zagadnienia optymalizacji oraz element losowości. Materiał genetyczny jest zawarty w populacji odpowiednio zakodowanych rozwiązań dopuszczalnych dla rozpatrywanego problemu optymalizacji. Można wyobrazić sobie wiele metod kodowania zadań, najbardziej popularne i efektywne jest kodowanie binarne [2]. Polega ono na zapisie rozwiązań problemu jako ciągów zero-jedynkowych. Nie każde jednak zadanie optymalizacji można zakodować w ten sposób. Przykładem mogą tu być właśnie rozpatrywane zadania szeregowania.

Algorytmy oparte na mechanizmach ewolucyjnych mają wiele zalet w porównaniu do tradycyjnych algorytmów optymalizacyjnych. Główną zaletą jest ich uniwersalność. Za pomocą tych samych lub bardzo zbliżonych i bardzo elementarnych operacji mutacji i krzyżowania oraz dziedziczenia można rozwiązać bardzo szeroką gamę problemów optymalizacyjnych.

Złożoność obliczeniowa takiego algorytmu jest niska i zależy głównie od implementacji problemu. Zależność od rozmiaru zadania (o ile występuje) jest rzędu wielomianowego.

Algorytm genetyczny jest zbieżny do rozwiązania optymalnego w sensie "prawie na pewno", co oznacza, że dla liczby iteracji dążącej do nieskończoności, odnajdziemy rozwiązanie optymalne. W praktyce jednak rozwiązania optymalne lub suboptymalne są znajdowane już po dość niskiej liczbie iteracji.

Algorytmy te umożliwiają optymalizację globalną, a nie tak jak większość podejść tradycyjnych, optymalizujących lokalnie. Jest to związane z prowadzeniem poszukiwań niezależnie przez całą populację rozwiązań, która wypełnia duży obszar.

Do uzyskania rozwiązania nie potrzebna jest znajomość pochodnych funkcji celu, nie konieczne są także żadne szczegółowe założenia co do samej funkcji celu, a w szczególności jej ciągłość i wypukłość. O wszelkich modyfikacjach znalezionych na jakimś etapie rozwiązań decydują czynniki losowe działające w całej populacji, a nie metody oparte na szczególnych właściwościach optymalizowanego procesu.

Zastosowanie algorytmów genetycznych napotyka jednak na pewne ograniczenia i trudności. Należą do nich problemy z odpowiednim zakodowaniem problemu oraz z dobraniem odpowiednich działań generujących kolejne populacje coraz lepszych rozwiązań, a także z odpowiednim doborem parametrów procesu, czyli np. prawdopodobieństw zachodzenia operacji rekombinacyjnych.

5. Algorytm genetyczny dla zadania szeregowania

Zagadnienie szeregowania zadań do wykonania na zbiorze procesorów jest problemem kombinacyjnym. Oznacza to, że różne rozwiązania otrzymamy przestawiając zadania między procesorami. Poddaje to myśl o możliwości wykorzystania w tym celu algorytmu genetycznego. Nie jest to myśl nowa, genetyczne podejście do rozwiązywania problemów związanych z procesami kolejkowymi można znaleźć np. w pracach [2] i [3]. W tej pracy zastosowano nieco inny sposób działania takiego algorytmu.

W algorytmie genetycznym każde możliwe rozwiązanie musi być w jakiś sposób zakodowane. Kod ten musi uwzględniać specyficzne cechy zadania i nie prowadzić do jego komplikacji. Dość intuicyjnym sposobem zakodowania procesu kolejkowego, jest przedstawienie go w postaci zestawu list zadań oczekujących w kolejce do danego procesora. Umożliwia on przeprowadzanie w prosty sposób wielu operacji na kolejkach do poszczególnych procesorów.

Tradycyjne operacje mutacji i krzyżowania są tu niedopuszczalne, gdyż w większości przypadków prowadziłyby do powstania rozwiązań niezgodnych z założeniami zadania. Wymiana dowolnych fragmentów kolejek między różnymi rozwiązaniami prowadziłoby do wielokrotnego pojawiania się tych samych zadań w jednych rozwiązaniach i znikania w innych. Podobne efekty dawałaby też mutacja, rozumiana jako losowe zastąpienie jednego zadania innym. Aby zapobiec takim zjawiskom należy zmodyfikować procesy modyfikacji rozwiązań.

Ponieważ każdy element populacji posiada pełny zestaw zadań do wykonania (pełny alfabet znaków kodujących problem), to każde rozwiązanie może wygenerować bez dostarczania nowych danych z zewnątrz (fragmentów kodu genetycznego od innych rozwiązań), przy użyciu określonych metod zmienności, nowe rozwiązanie. Oznacza to, że można zrezygnować z wymiany części kodu genetycznego między „osobnikami" populacji rozwiązań, a ograniczyć się do wymiany fragmentów kodu między segmentami wewnątrz osobnika. Segmenty te określają kolejkę zadań do jednego stanowiska obsługi. W ten sposób w kolejnych iteracjach algorytmu powstają wciąż nowe osobniki o zupełnie nowych cechach (wartościach funkcji celu). Operację tę można nazwać autokrzyżowaniem.

Należy tu zwrócić uwagę na fakt występowania w przyrodzie pewnych analogii do takiego sposobu wymiany materiału genetycznego. Istnieją przecież organizmy obojnacze u których wymiana materiału genetycznego następuje w obrębie tego samego zestawu chromosomów.

Wybór segmentów, między którymi przeprowadzana jest wymiana można przeprowadzać losowo lub też wprowadzić tu element deterministyczny, opierając się na właściwościach problemu optymalizacji. Wiadomo, że poprawę bieżącego rozwiązania można otrzymać przez skrócenie najdłuższej kolejki. Dlatego też dobrze jest przeprowadzać wymianę między najdłuższym i najkrótszym segmentem (co do czasu wykonywania, a nie co do liczby zadań w kolejce). Prowadzi to do najszybszego poszukiwania lepszych rozwiązań. Można wobec tego wprowadzić dwa rodzaje autokrzyżowania: losowe i deterministyczne. Autokrzyżowanie losowe umożliwia zachowanie możliwości wpływania na wszystkie segmenty rozwiązań (zwiększa zmienność populacji), autokrzyżowanie deterministyczne prowadzi do przyspieszenia efektywności poszukiwań coraz lepszych rozwiązań. Punkt wymiany w obu wariantach autokrzyżowania jest wybierany losowo z tym, że dla dłuższej kolejki dużo lepsze efekty da rozcięcie kolejki bliżej końca. Dlatego też punkt wymiany jest w niej losowany między środkiem a końcem (co do liczby zadań w danej kolejce). W kolejce krótszej punkt wymiany jest wybierany losowo w dowolnym jej miejscu.

Mutacja, polegająca na losowej zmianie jednego z zadań w kolejce na inne również prowadziłaby do rozwiązań niedopuszczalnych. Jedną z możliwości zapobieżenia temu jest mutacja polegająca na zamianie dwóch zadań miejscami w obrębie jednego osobnika.

W algorytmie zastosowano również inwersję, czyli odwracanie kolejności zadań w kolejce do jednego ze stanowisk obsługi. Stanowisko jak i punkty dla których następuje zmiana kolejności są wybierane losowo. Inwersja nie zmienia wartości funkcji celu dla danego uszeregowania, zwiększa jednak zmienność populacji i jest istotnym czynnikiem ułatwiającym znajdowanie coraz lepszych rozwiązań.

W symulacjach zastosowano dwie metody reprodukcji: metodą ruletki i metodą wartości oczekiwanej [2]. Znacznie szybsze postępy algorytmu zaobserwowano dla drugiej metody. Metoda ta preferuje lepiej dopasowane osobniki i praktycznie eliminuje wszystkie o słabym dopasowaniu. Jest to ważne z uwagi na dużą wariancję rozkładu długości uszeregowania dla całej populacji. Rozkład ten nie charakteryzuje się symetrią względem wartości średniej. Osobniki najgorsze dają rozwiązania kilka razy dłuższe od średniego, gdy rozwiązania najlepsze są od niego tylko o kilka procent lepsze. Bardzo dobre wyniki daje w tej sytuacji stosowanie nieliniowej funkcji reprodukcji (z zastosowaniem metody wartości oczekiwanej). Liczebność otrzymanej populacji jest oczywiście skalowana tak, aby jej wielkość się nie zmieniała. Funkcja reprodukcji ma postać:

$$n = \text{round}(\text{rand} * (C_{sr} - C^*)/(C_i - C^*))$$

gdzie:

n - liczba osobników potomnych;
C^* - oszacowanie od dołu długości uszeregowania;
C_i - długość uszeregowania dla i-tego osobnika populacji;
$C_{śr}$ - średnia wartość długości uszeregowania dla populacji;
rand - funkcja generująca losowo liczbę z przedziału $\langle 0,1 \rangle$;
round - zaokrąglanie wyniku do najbliższej liczby całkowitej.

Algorytm genetyczny, umożliwiający poszukiwanie rozwiązań zarówno dla przypadku procesorów jednakowych, jak i różniących się prędkością. Szczegółowe wyniki symulacji przedstawiono w dalszej części pracy. W tym miejscu chciałbym jedynie nadmienić, że mimo że otrzymywane rezultaty były bardzo dobre, to jednak ich znalezienie wymagało dość znacznego nakładu obliczeń. Tradycyjne algorytmy przybliżone znajdowały rozwiązania, co prawda gorsze, lecz w dużo krótszym czasie. Stąd powstał pomysł wykorzystania nienajlepszych lecz szybkich rozwiązań generowanych przez algorytmy przybliżone i wykorzystania ich jako punktu startowego do poszukiwań dla algorytmu genetycznego. Dla konkretnego zadania optymalizacyjnego algorytm tradycyjny znajdzie oczywiście tylko jedno rozwiązanie, nie ma tu możliwości wygenerowania całej populacji rozwiązań. Jednakże rozwiązanie to można poddać odpowiednią liczbę razy niezależnie operacjom autokrzyżowania, przestawiania i mutacji tak, że powstanie cała populacja rozwiązań o nieco zaburzonych parametrach wokół rezultatu poszukiwań algorytmem tradycyjnym. Tak wystartowana symulacja praktycznie w każdym przypadku i dość szybko potrafiła znacznie poprawić odnalezione wcześniej uszeregowanie.

6. Rezultaty przeprowadzonych symulacji

Wykres 3 pokazuje działanie przedstawionego wyżej algorytmu genetycznego dla kilku różnych liczb stanowisk obsługi i ilości zadań w kolejce. Oś rzędnych na wykresie reprezentuje

kolejny numer iteracji, oś odciętych stosunek C_{min}/C^*, gdzie C_{min} oznacza najlepszego reprezentanta populacji rozwiązań, C^* zaś oszacowanie od dołu rozwiązania problemu. Przyjęto różne prędkości działania poszczególnych stanowisk (procesory jednorodne). Pozostałe parametry symulacji: populacja 30 rozwiązań, prawdopodobieństwo mutacji $P_{mut}=0,1$, prawdopodobieństwo krzyżowania $P_{cross}=0,8$, prawdopodobieństwo inwersji $P_{inv}=0,3$. W symulacjach zastosowano obie opisane wyżej wersje krzyżowania (losowano sam fakt wystąpienia krzyżowania a następnie jego rodzaj): kolejki najdłuższej z najkrótszą P=0,8 i wybranych losowo z P=0,2. Populacja startowa była generowana losowo. Czasy wykonania zadań generowane były z rozkładem jednostajnym z przedziału {20..50}.

Na wykresie widoczne jest znaczne wydłużenie czasu poszukiwania dla bardzo dużych problemów (160 procesorów i 15000 zadań). Dla problemów średnich i małych bardzo dobre rozwiązania są znajdowane bardzo szybko, potrzebnych jest ok. 100 iteracji. Czas trwania jednej iteracji opisanego algorytmu genetycznego jest proporcjonalny do liczby zadań w problemie oraz liczności populacji. Liczba iteracji potrzebna do uzyskania odpowiedniej jakości rozwiązania jest trudna do oszacowania. Na pewno zależy ona od liczby procesorów, a w nieco mniejszym stopniu od liczby zadań w kolejce. Przykładowe wartości uzyskane symulacyjnie przedstawia tabela 1. Zestawiono tam wyniki doświadczenia w którym mierzona była liczba iteracji potrzebna do osiągnięcia rozwiązania C_{min} o 1% gorszego od C^*. Nie zawsze możliwe było osiągnięcie takiej dokładności, wynikało to jednakże nie z ograniczonych możliwości algorytmu lecz najprawdopodobniej z powodu istnienia rozwiązania optymalnego różniącego się od C^* o więcej niż 1%. Analiza wyników tabeli prowadzi do zaskakującego wniosku - dla małej liczby zadań należy przeprowadzić więcej iteracji niż dla dużej (przy danej liczbie procesorów). Wynika to najprawdopodobniej z faktu, że przy małej liczbie zadań istnieje mało odpowiednio dobrych uszeregowań, wobec czego trudniej algorytmowi w nie trafić. Zjawisko to staje się szczególnie widoczne dla wartości stosunku $n/m<10$. Wyraźnie widoczna jest też zależność liczby iteracji od liczby stanowisk obsługi w rozpatrywanym problemie. Wyniki przedstawione w tabeli pokazują wartości średnie uzyskane w 10 próbach.

m \ n	10	100	1000	10000
2	2,2	1,4	3	1,6
4	-	14,2	13,7	14,9
8	-	39,6	41,4	44,1
16	-	221,2	146,2	113,9
32	-	-	348,5	331,2
64	-	-	819,2	699
128	-	-	2417,8	1482,6

Tabela 1. Średnia liczba iteracji algorytmu genetycznego, konieczna do uzyskania dokładności ok. 1%.

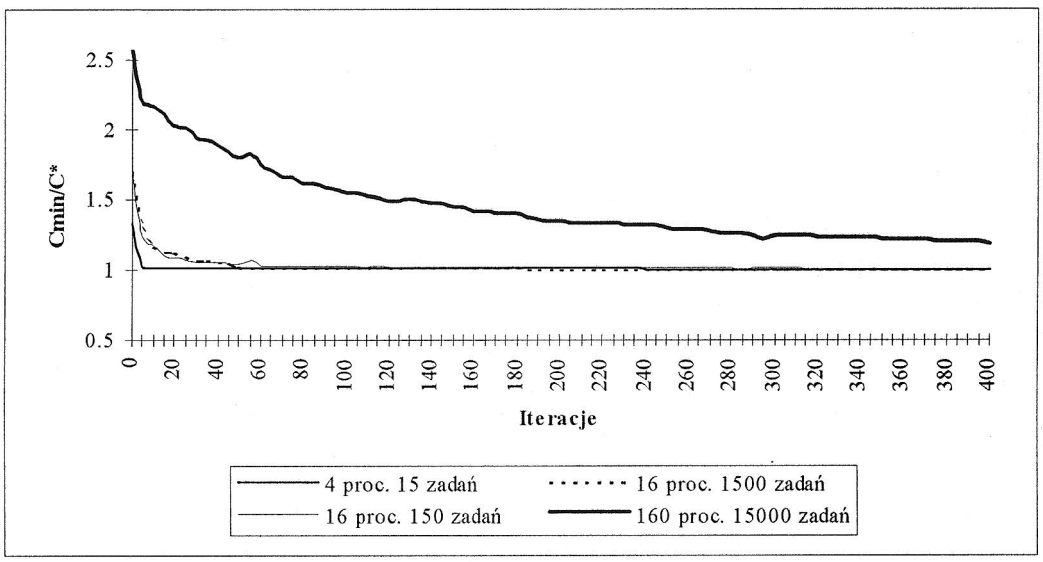
Dla bardzo dużych zadań użyteczność algorytmu genetycznego jest nieduża. Znacznie lepsze wyniki daje wtedy deterministyczny algorytm szeregowania, działający znacznie szybciej i z większą dokładnością. Dla mniejszych zadań algorytm deterministyczny działa również nieco szybciej niż genetyczny, lecz daje znacznie gorsze rezultaty. Sposobem na połączenie zalet obu algorytmów może być algorytm hybrydowy, łączący w sobie cechy obu podejść.

Algorytm hybrydowy polega na wygenerowaniu startowego rozwiązania przy użyciu algorytmu deterministycznego, a następnie wygenerowaniu startowej populacji dla algorytmu

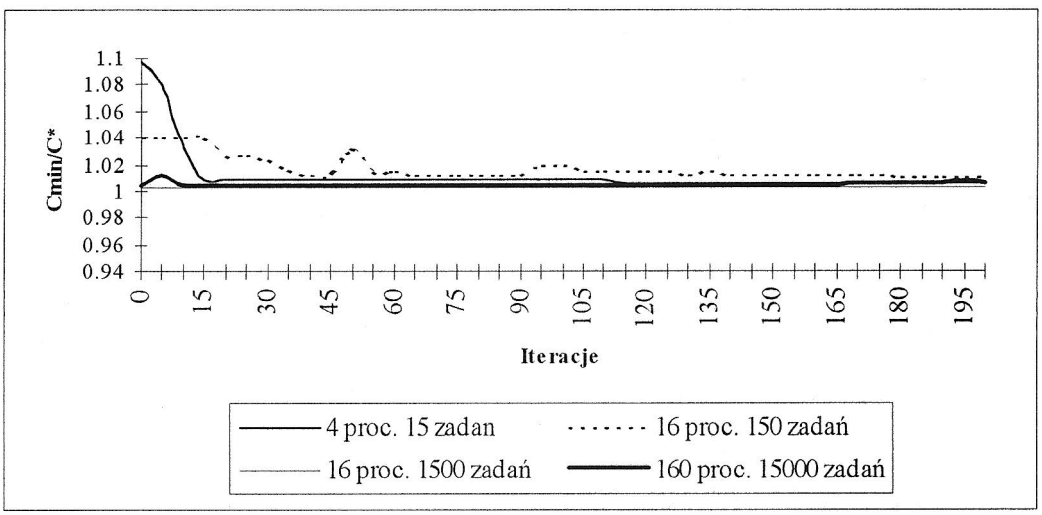
genetycznego przez „rozmnożenie" tego rozwiązania (zakodowanego) i poddanie każdego osobnika wstępnym operacjom mutacji, krzyżowania i inwersji. Następnie następuje faza tworzenia populacji potomnej.

Wszystkie te operacje działają tak samo jak w standardowym algorytmie opisanym wcześniej. Działanie algorytmu hybrydowego dla parametrów takich samych jak w przypadku poprzednim przedstawia wykres 4.

Wykres 4 potwierdza wyrażone wcześniej zdanie, że klasyczny przybliżony algorytm szeregowania daje tym lepsze rozwiązania im większy problem ma do rozwiązania. Dla zadań małych i średnich parametry rozwiązań nie są zbyt dobre. Oszacowania dokładności przybliżonych algorytmów szeregujących dla zadań z jednakowymi lub jednorodnymi procesorami można znaleźć w pracy [1].



Wykres 3. Wyniki działania algorytmu genetycznego minimalizującego długość uszeregowania.



Wykres 4. Przykłady działania algorytmu hybrydowego.

7. Podsumowanie

Przedstawiony w tej pracy sposób wykorzystania algorytmów genetycznych do optymalizacji procesów kolejkowych nie jest ograniczony jedynie do opisanych wyżej zagadnień. Można zastosować go po niewielkich modyfikacjach również do rozwiązywania bardziej skomplikowanych problemów szeregowania zadań z ograniczeniami np. typu kolejnościowego (ograniczenia można uwzględnić w funkcji dopasowania). Możliwe jest także zastosowanie go w innego typu zadaniach, gdzie proste operacje krzyżowania i mutacji między osobnikami w populacji prowadzą do rozwiązań niedopuszczalnych. Przykładem takiego problemu może być zadanie komiwojażera.

8. Literatura

[1] Jacek Błażewicz, Wojciech Cellary, Roman Słowiński, Jan Węglarz, *„Badania operacyjne dla informatyków"*, Wydawnictwo Naukowo Techniczne, Warszawa 1983.
[2] David E. Goldberg, *„Algorytmy genetyczne i ich zastosowania"* , Wydawnictwo Naukowo Techniczne, Warszawa 1995.
[3] Davis L.: *„Job shop scheduling with genetic algorithms"*, Proceedings of an International Conference on Genetic Algorithms and Their Applications, 136-140.
[4] Eugeniusz Toczyłowski, *„Niektóre metody strukturalne optymalizacji do sterowania w dyskretnych systemach wytwarzania"*, Wydawnictwo Naukowo Techniczne, Warszawa 1989.