AUTOMATYKA • 2011 • Tom 15 • Zeszyt 2

Mieczysław Drabowski*, Edward Wantuch**

Metody sztucznej inteligencji w syntezie systemów typu kompleks zasobów i operacji

1. Wprowadzenie

Specyfikacja opisująca projektowany system typu kompleks zasobów i operacji może być definiowana, jako zbiór zadań (operacji, procesów, funkcji), które należy wykonać na wybranych (ze znanej puli) dostępnych zasobach optymalizując w procesie wysokopoziomowej syntezy koszt, czas realizacji, pobór mocy jak również niezawodność (wiarygodność) systemu [1, 10].

Podstawowym problemem w syntezie tak określanego systemu jest podział funkcji na te implementowane sprzętowo oraz programowo. Podział ten jest ważny, ponieważ każdy system komputerowy jak również współczesny system wytwarzania jest zrealizowany w wyniku implementacji sprzętowej niektórych jego zadań.

W dotychczasowych metodach syntezy, części sprzętowe i programowe były opracowywane i optymalizowane sekwencyjnie, a następnie łączone w procesie tzw. co-syntezy (ang. *concurrent synthesis*), co zwiększało koszt i zmniejszało szybkość realizacji oraz niezawodność końcowego produktu. Celem podziału funkcji systemu jest określenie, jeszcze przed opracowywaniem szczegółów realizacyjnych, jakie zasoby (sprzętowe i programowe) są wymagane do pełnej realizacji systemu. Równie ważnym zagadnieniem syntezy systemów jest problem szeregowania zadań, który występuje przy opracowywaniu procedur operacyjnych odpowiedzialnych za sterowanie rozdziałem (alokacją) zadań i zasobów.

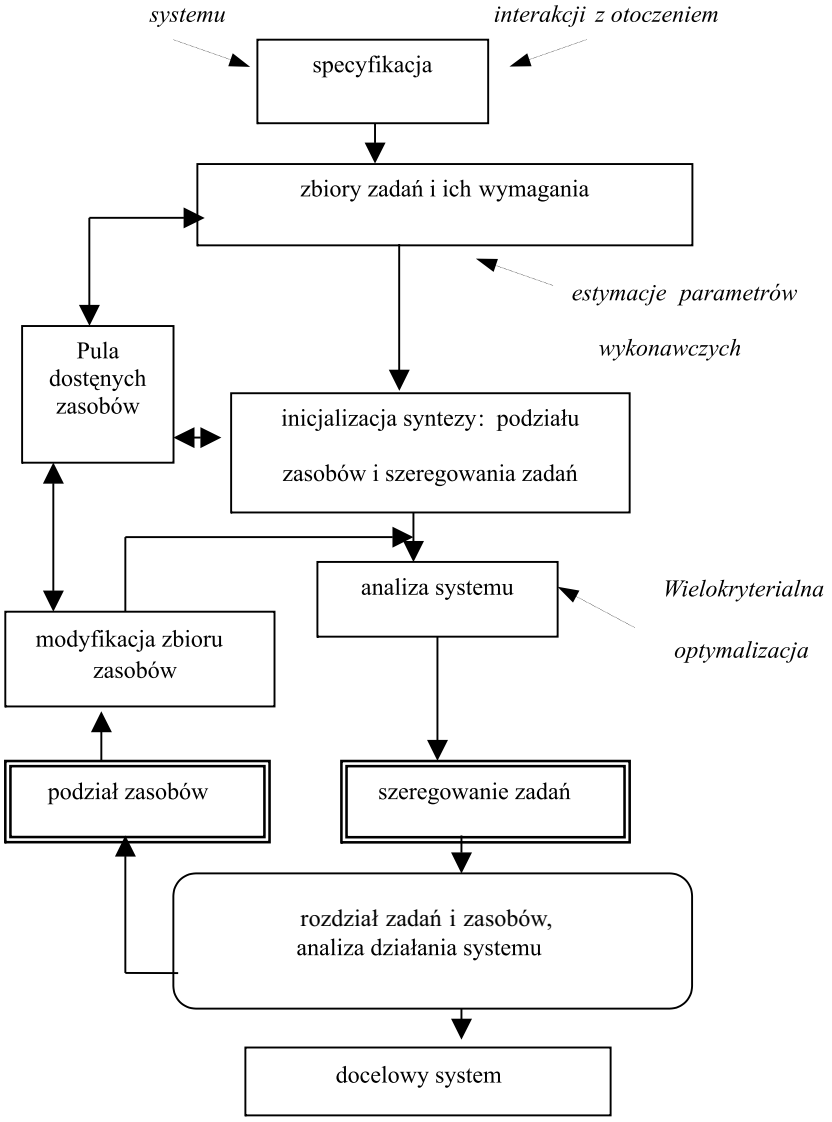
Celem niniejszych badań jest ocena metody spójnego (koherentnego) rozwiązywania problemów podziału zasobów i problemów szeregowania zadań według modelu i koncepcji zaprezentowanej w [2], umożliwiającej synergiczne projektowanie sprzętu oraz oprogramowania sterującego działaniem projektowanego systemu. Takie podejście to tzw. par-synteza (paralelna synteza). Badania przedstawiają wyniki wybranych reprezentatywnych

* Politechnika Krakowska, Wydział Inżynierii Elektrycznej i Komputerowej, Katedra Informatyki Technicznej

** AGH Akademia Górniczo-Hutnicza, Wydział Inżynierii Mechanicznej i Robotyki, Katedra Systemów Wytwarzania

eksperymentów obliczeniowych dla różnych instancji problemów par-syntezy, wykazujących prawidłowość koncepcji koherentnej syntezy i pokazujących metody rozwiązania tych problemów.

W procedurze badawczej zostały zastosowane wybrane podejścia metaheurystyczne (inteligencji obliczeniowej), takie jak: genetyczne z symulowanym wyżarzaniem, poszukiwania tabu, neuronowe i kolonii mrówek.



Rys. 1. Proces par-syntezy systemów komputerowych

Schemat koherentnego procesu syntezy systemów komputerowych realizowanego przez te heurystyki został przedstawiony na rysunku 1 i składa się z następujących etapów:

1. Specyfikacja wymagań dla projektowanego systemu i jego interakcji z otoczeniem. Zdefiniowanie funkcjonalności systemu.
2. Określenie zadań realizujących w pełni funkcje systemu. Estymacje parametrów wykonawczych zawartych w tych zadaniach oraz procedur na dostępnych zasobach (np. czasów wykonania, wymagań na przestrzeń pamięci, definicje zależności między procedurami, w tym czasowe). Określenie ograniczeń i wymagań krytycznych z punktu widzenia realizowalności.
3. Przyjęcie początkowych (inicjujących): zbioru zasobów oraz uszeregowania zadań z puli zasobów i uszeregowań, a także z bazy danych historycznych pamiętanych z syntezy systemów w przeszłości; początkowo przyjęty zbiór zasobów i uszeregowanie powinny realizować wszystkie wymagania, lecz niekoniecznie optymalnie.
4. Szeregowanie zadań oraz rozdział zadań i zasobów – zadań ze specyfikacji, a także zasobów aktualnie wybranych i przyporządkowanych do zadań.
5. Ocena czasu wykonania, poboru mocy i kosztu systemu; optymalizacja wielokryterialna.
6. W wyniku tej oceny następuje modyfikacja zbioru zasobów i ponowny podział oraz powrót do punktu 4 lub uzyskanie satysfakcjonującego rozwiązania.

Obliczenia iteracyjne są realizowane do momentu osiągnięcia wystarczająco dobrych rezultatów – osiągnięcie docelowego optymalnego (lub suboptymalnego) systemu, tzn. jego struktury i uszeregowania.

System taki powinien być szybki, tani i małej mocy. Można zauważyć w tym procesie równoległość w realizacji szeregowania zadań i podziału zasobów (stąd termin par-synteza).

2. Adaptacje metaheurystyk

2.1. Metoda ewolucyjna

Dla przeciwdziałania konwergencji rozwiązań, w algorytmie ewolucyjnym zastosowano struktury danych zapewniające zachowanie lokalności cech opisanych przez wektory wartości występujących w chromosomach. Lokalność interpretowana jest, jako odwrotność odległości wektorów wewnątrz n-wymiarowej hipersfery [3], wówczas operatory krzyżowania i mutacji nie są operacjami wymiany danych pomiędzy wektorami, ale pomiędzy fragmentami hipersfer. Dzięki takiemu podejściu, niewielkie zmiany w chromosomie odpowiadają niewielkim zmianom w rozwiązaniu opisywanym przez ten chromosom. W prezentowanym rozwiązaniu występują dwie hipersfery: zadań i zasobów.

Rozwiązania posiadające taką samą alokację zasobów tworzą tzw. skupiska. Wprowadzenie skupisk rozwiązań odseparowuje od siebie rozwiązania o różnych alokacjach. Rozwiązania takie ewoluują oddzielnie, co zabezpiecza operację krzyżowania przed wytwarzaniem wadliwych rozwiązań. Nie dochodzi również do sytuacji, w której zadanie zostanie

przypisane do niezaalokowanego zasobu. Skupiska rozwiązań opisują struktury projektowanego systemu (stanowią je zasoby przeznaczone do alokacji zadań). Rozwiązania są odwzorowaniem alokowanych zadań do zasobów i uszeregowań zadań. Podczas ewolucji zachodzą dwa typy operacji genetycznych (krzyżowania i mutacji), które odbywają się na dwóch poziomach (skupiska i rozwiązania).

Utworzona zostaje populacja, której parametrami są: liczba skupisk, liczba rozwiązań w skupiskach, graf zadań oraz biblioteka zasobów. Dla potrzeb syntezy są określane kryteria optymalizacji oraz kryterium zakończenia iteracji algorytmu, jeśli nie nastąpiła poprawa rozwiązania. Jako kryteria optymalizacji przyjęto maksymalną liczbę pokoleń ewoluujących rozwiązań wewnątrz skupisk, przy maksymalnie ograniczeniach liczby zasobów, ich całkowitego kosztu, całkowitego czas realizacji wszystkich zadań, poboru mocy przez realizowany system oraz opcjonalnie wielkości listy najlepszych i niezdominowanych osobników [4].

2.2. Metoda tabu-search

W algorytmie Tabu-search do koherentnej optymalizacji podziału zasobów i szeregowania zadań rozwiązanie startowe generowane jest losowo lub za pomocą innej heurystyki. Otrzymywane rozwiązania początkowe nie są optymalne, ale spełniają wszystkie wymagania sprecyzowane w danych wejściowych i dalszą optymalizację przeprowadza algorytm Tabu-Search. Podstawową ideą tego algorytmu jest przeszukiwanie przestrzeni wszystkich możliwych rozwiązań za pomocą procedury definiującej sekwencje tzw. ruchów (*move*). W sekwencji ruchów mogą istnieć ruchy tabu – ruchy niedozwolone; algorytm unika oscylacji wokół optimum lokalnego dzięki przechowywaniu informacji o sprawdzonych już rozwiązaniach w postaci listy tabu. W algorytmie reprezentującym podejście koherentne do problemów optymalizacji szeregowania zadań i podziału zasobów, ruch został określony, jako przeniesienie określonej części zadania (lub całego zadania), wykonywanego na jednym z procesorów, na inny procesor i inną pozycję w uszeregowaniu [5]. Zadania wykonywane na wymienianym procesorze zostają przeniesione na nowy procesor pobrany ze zbioru zasobów lub na inne aktywne procesory. Taka sytuacja może nastąpić na przykład, gdy zadania są zależne, a nowy procesor jest wolniejszy od poprzedniego lub dla przypadku problemów z zasobami dodatkowymi. W przypadku koherentnego podejścia do szeregowania zadań i podziału zasobów, otoczenie funkcyjne, jak i sąsiedztwo są zbiorami podwójnymi, oddzielne – dla szeregowania zadań oraz podziału zasobów. Tworzenie nowych ruchów częściowo odbywa się w sposób losowy. Najpierw generowane są ruchy dla szeregowania zadań, po ich wykonaniu i ewentualnym powrocie do rozwiązania najlepszego, dotąd znalezionego, następuje generowanie ruchu dla problemu podziału zasobów. Tego rodzaju rozwiązanie pozwala na spójny podział zasobów i sprawdzanie różnych uszeregowań zadań dla tych podziałów zasobów.

Na pamięć krótkoterminową składają się wszystkie ostatnio wykonywane ruchy. Oprócz modyfikacji opisanych w poprzednich rozdziałach, w przypadku metody Tabu-Search

wprowadzono nowe. Lista tabu związana z podziałem zasobów, posiada mechanizmy pozwalające na dowiązanie do każdego z elementów listy ruchów dla podziału zasobów oraz listy ruchów dla szeregowania zadań. Późniejsze przeszukiwanie list ogranicza się tylko do odszukania ruchu dla podziału i dla tego ruchu sprawdzenia istnienia na dowiązanej do niego liście, ruchu dla szeregowania. Jest to mechanizm pozwalający na sprawne i efektywne pamiętanie ruchów oraz ich przeszukiwanie.

Jako pamięć długoterminową została użyta lista wygenerowanych rozwiązań. Informacje uzyskiwane na podstawie wartości tych zmiennych są, podobnie jak dla poprzednich algorytmów, następujące: całkowita liczba iteracji wykonanych od początku działania algorytmu, liczba iteracji bez uzyskania lepszego rozwiązania, liczba dotychczasowych powrotów do rozwiązania początkowego, aktualna wartość parametru określającego urozmaicenie w przeszukiwaniu otoczenia. Rozpatrywanymi kryteriami są jednocześnie całkowity koszt, pobierana moc i czas wykonania wszytskich zadań (tzn. długość uszeregowania $\boldsymbol{C_{max}}$ zadań).

2.3. Metoda neuronowa

Punktem wyjścia do sformułowania modelu sieci neuronowej dla rozwiązywania problemów szeregowania zadań i podziału zasobów jest problem zaspokajania ograniczeń (CSP – *Constraint Satisfaction Problem*) [6]. CSP to problem optymalizacyjny, w którym występuje określony zbiór zmiennych, zbiory ich możliwych wartości oraz ograniczenia narzucone na wartości tych zmiennych. W oparciu o założenia tego problemu model sieci dla naszych problemów charakteryzuje się następującymi własnościami:

- sieć neuronowa składa się z elementów, z których każdy odpowiada innej zmiennej,
- w każdym elemencie jest tyle neuronów, ile możliwych wartości danej zmiennej,
- nadanie zmiennej określonej wartości polega na włączeniu odpowiedniego neuronu (neuronów) i wyłączeniu pozostałych w części odpowiadającej tej zmiennej,
- włączenie neuronu to nadanie jego wyjściu wartości „1”,
- wyłączenie neuronu to nadanie jego wyjściu wartości „0”,
- ograniczenia w sieci wprowadza się przez dodanie połączenia o ujemnej wadze („–1”) pomiędzy neuronami, symbolizującymi wartości zmiennych, które nie mogą wystąpić jednocześnie,
- w sieci mogą występować dodatkowe neurony „jedynki”, które na stałe są włączone.

Każdy neuron ma tablicę połączeń, a każde połączenie zawiera swoją wagę i wskaźnik do podłączonego neuronu. Cechą charakterystyczną tej sieci jest to, że połączeń między neuronami jest wiele, ale nigdy nie są to połączenia do wszystkich neuronów [7]. Wynika to z faktu, że połączenia między neuronami istnieją tylko tam, gdzie narzucone są jakiekolwiek ograniczenia. Ograniczenia występujące w omawianym modelu sieci, mogą być typu: zasobowego, czasowego lub kolejnościowego. Po wprowadzeniu danych wejściowych (specyfikacji systemu) algorytm konstruuje sieć neuronową, której złożoność i liczba

wchodzących w jej skład neuronów zależy od wielkości i złożoności instancji problemu. Część sieci przydzielona danemu zadaniu nazwiemy obszarem. Do sieci wprowadzane są ograniczenia poprzez realizację połączeń, które jak już wspomniano, występują tylko pomiędzy neuronami odpowiadającymi tym wartościom zmiennych, które nie mogą wystąpić jednocześnie. Działanie algorytmu polega na włączaniu odpowiednich neuronów w każdym obszarze sieci tak, aby spełnić przy tym jak najwięcej ograniczeń narzuconych przez dane wejściowe. Przebieg algorytmu jest następujący:

1. Nadanie kolejnym zmiennym losowych wartości.
2. Relaksacja sieci:
   – obliczenie sumy ważonej wejść wszystkich neuronów,
   – włączenie neuronu o największej wartości na wejściu,
   – powrót do relaksacji lub w przypadku braku zmian wyjście z relaksacji.
3. Jeśli między włączonymi neuronami istnieją połączenia (ograniczenia) to każdą wagę między dwoma włączonymi neuronami zostaje pomniejszonao 1 i algorytm wraca do relaksacji.

Algorytm rozpoczyna pracę od nadania wszystkim połączeniom wagi o ujemnej wartości „–1”, następnie generowane jest rozwiązanie początkowe. Powstaje ono przez nadanie kolejnym zmiennym – losowych wartości. Proces ten odbywa się w określony sposób: dla każdego zadania, losowo włącza się tyle neuronów, ile wymaga dane zadanie do swojego wykonania. Pozostałe neurony w części odpowiadającej danemu zadaniu są wyłączane. Otrzymany w ten sposób wynik nie jest jeszcze rozwiązaniem, gdyż istnieje w nim wiele sprzeczności, symbolizowanych włączeniem neuronów, między którymi istnieją połączenia.

Dlatego następnym krokiem algorytmu jest proces relaksacji, którego celem jest osiągnięcie stanu równowagi, w którym uzyskamy „zaspokojenie” jak największej liczby ograniczeń (ang. backtracking). Chodzi mianowicie o to, aby uzyskać wynik, w którym liczba sytuacji, w których włączone są dwa neurony, między którymi jest połączenie o ujemnej wadze, jest najmniejsza.

2.4. Metoda kolonii mrówek

Algorytm kolonii mrówek (ACO – *Ant Colony Optimization*) jest algorytmem heurystycznym wykorzystującym ideę agentów (tj. mrówek) w sposób imitujący ich rzeczywiste zachowanie [8]. Na podstawie kilku informacji (odległość, liczba feromonu na ścieżkach itp.) mrówki oceniają, „jakość” ścieżek i wybierają je z pewnym prawdopodobieństwem, (które jest tym większe, im lepsza jest, jakość danej ścieżki). Po przebyciu całej ścieżki, od źródła do celu, mrówki uczą się od siebie nawzajem poprzez odkładanie na ścieżce warstwy feromonu zależnej od rozwiązania, które zostało obrane przez konkretnego agenta: im lepsze rozwiązanie, tym większa ilość feromonu zostaje odłożona. Feromon następnie „ulatnia się”, by umożliwić zmianę trasy obranej przez mrówki i zarzucenie gorszych ścieżek, któ-

rymi wcześniej się przemieszczały. Rezultatem takiego działania algorytmu nie zawsze jest samo rozwiązanie. Niejednokrotnie jest nim ślad, z którego rozwiązanie się wywodzi. Pozwala to wówczas nie tylko na analizę pojedynczego rozwiązania, ale permutacji generujących rozwiązania inne, choć bazujące na takim samym podziale (np. zadania zostają uszeregowane w innej kolejności, choć w dalszym ciągu przypisywane są do tych samych procesorów). Właśnie takie podejście ma istotne zastosowanie do rozwiązywania problemów takich, jak synteza, gdzie nie tylko sam podział zadań ma znaczenie, ale także i kolejność ich uszeregowania.

W analizie problemu algorytm ACO potrzebuje zdefiniowanych dwóch parametrów:

- liczby agentów (mrówek) w kolonii – parametr *a,*
- współczynnika waporyzacji („ulatniania się") feromonu, którego wartość powinna zawierać się w przedziale otwartym (0; 1) – parametr γ.

Dobór tych parametrów jest istotny:

- dla zbyt dużej liczby agentów pojedynczy cykl algorytmu będzie trwał bardzo długo, a wartości zapamiętywane w tablicy („poziomy feromonu") w wyniku sumowań będą wyznaczały stosunkowo słabe rozwiązania;
- jednak, przy niewielkiej liczbie agentów większość ścieżek nie będzie pokryta i w rezultacie najlepsze rozwiązanie może długo pozostać nieodkryte.

Podobnie dla wskaźnika waporyzacji:

- wartość zbyt mała spowoduje, iż mrówki będą szybko „zapominały" dobre rozwiązania i w rezultacie może w krótkim czasie dojść do tzw. *stagnacji* (algorytm poprzestanie na jednym rozwiązaniu, które nie musi być najlepszym);
- zbyt duża wartość dla tego współczynnika powoduje natomiast, iż mrówki nie przestają analizować rozwiązań „słabych"; ponadto nowo odnalezione rozwiązania mogą nie zostać przeforsowane, jeżeli czas, który upłynął od ostatniego znalezionego rozwiązania będzie odpowiednio długi (wartości feromonu zapamiętane w tablicy będą zbyt duże).

Algorytm ACO dodatkowo definiuje jeszcze dwa parametry pozwalające na róznicowanie pomiędzy:

- α – ilością feromonu na ścieżce, a
- β – "jakością" kolejnego kroku.

Parametry te są dobierane stosownie do potrzeby konkretnej instancji problemu. W ten sposób dla parametrów:

- $\alpha > \beta$ uzyskuje się większy priorytet na wybór ścieżki bardziej uczęszczanej,
- $\alpha < \beta$ uzyskuje się większy priorytet na ścieżkę oferującą lepsze rozwiązanie,
- $\alpha = \beta$ otrzymamy zrównoważoną zależność między jakością ścieżki a stopniem jej uczęszczania,

- $\alpha = 0$ otrzymamy algorytm heurystyczny opierający się wyłącznie na jakości przejścia między kolejnymi punktami (ignorancja poziomu feromonu na ścieżce),
- $\beta = 0$ otrzymamy algorytm bazujący wyłącznie na ilości feromonu (a zatem wskaźnika uczęszczania ścieżki),
- $\alpha = \beta = 0$ otrzymamy algorytm dokonujący podziału w sposób równomierny i niezależny od ilości feromonu lub jakości rozwiązania.

Mając dany zbiór sąsiedztwa N danego punktu i, ilość feromonu na ścieżce τ oraz jakość przejścia z punktu i do punktu j, jako element tablicy η możemy przedstawić prawdopodobieństwo przejścia z punktu i do j, w postaci formuły 2.1.

$$p_{ij}^{k} = \begin{cases} \dfrac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} [\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}} & \text{jeśli} \quad j \in N_i^k \\ 0 & \text{w przeciwnym razie} \end{cases}$$

Formuła 2.1. Ocena jakości kolejnego k kroku w algorytmie ACO

W metodzie prezentowanej przez algorytm ACO wykorzystuje się agentów do wyszukiwania trzech informacji [9]:

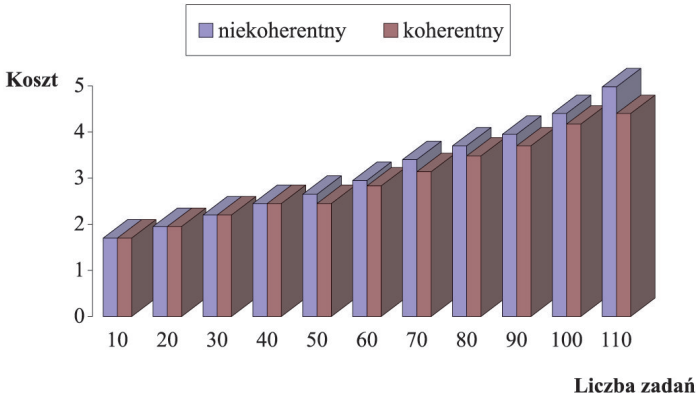
- najlepszego / najkorzystniejszego podziału zadań między procesorami,
- najlepszej kolejności wyboru kolejnych zadań,
- wyszukiwania najlepszego osiągalnego rozwiązania dla danego rozkładu.

Agenci (mrówki) wyszukują rozwiązania będące zestawem wynikającym z pierwszych dwóch celów (dają one w rezultacie rozwiązanie jednoznaczne).

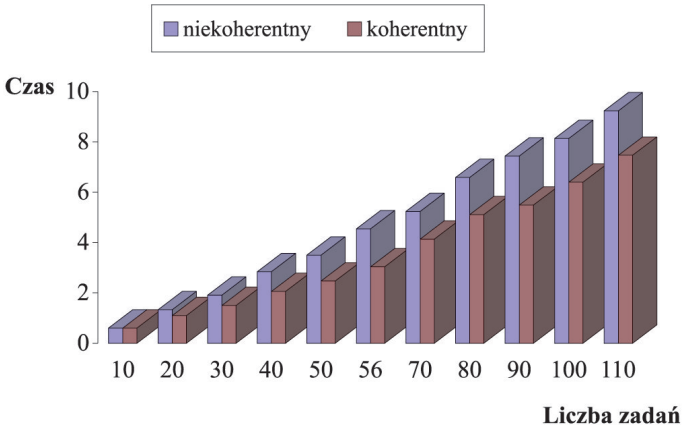
3. Wyniki przykładowych obliczeń

3.1. Wyniki obliczeń procesów co-syntezy i par-syntezy

Celem przeprowadzonego eksperymentu obliczeniowego było porównanie syntezy koherentnej i niekoherentnej, a uzyskane wyniki zaprezentowano na przykładzie metody ewolucyjnej (dla pozostałych metod wyniki są podobne). Na podstawie wyników przedstwionych na rysunku 2 można wnioskować, że par-synteza daje lepsze rezultaty generując struktury tańsze (powyżej 10%); przy tym zysk relatywnie rośnie dla większej liczby zadań. Dane przedstawione na rysunku 3 wskazują, że w przypadku wyników uzyskiwanych dla par-syntezy następuje poprawa otrzymywanych wyników – dla kryterium minimalizacji czasu realizacji jest to ponad 20%, przy tym zysk również relatywnie rośnie dla większej liczby zadań.



Rys. 2. Porównanie par-syntezy (podejście koheretne) i co-syntezy (podejście niekoherentne) dla kryterium minimalizacji kosztu systemu



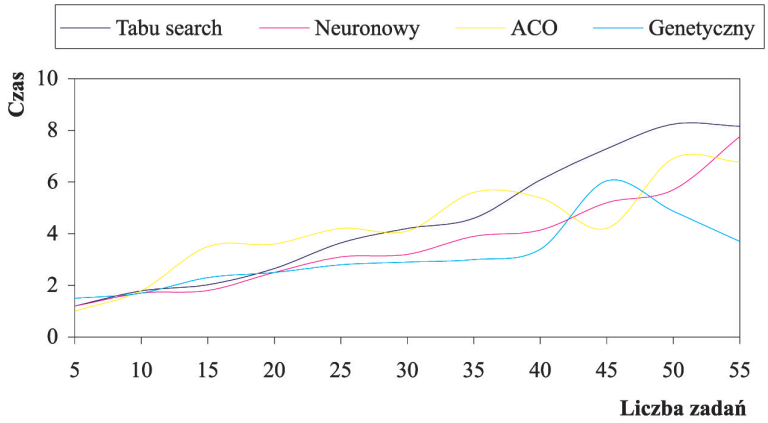
Rys. 3. Porównanie par-syntezy (podejście koheretne) i co-syntezy (podejście niekoherentne) dla kryterium minimalizacji czasu

3.2. Wybrane wyniki porównania wszystkich metod

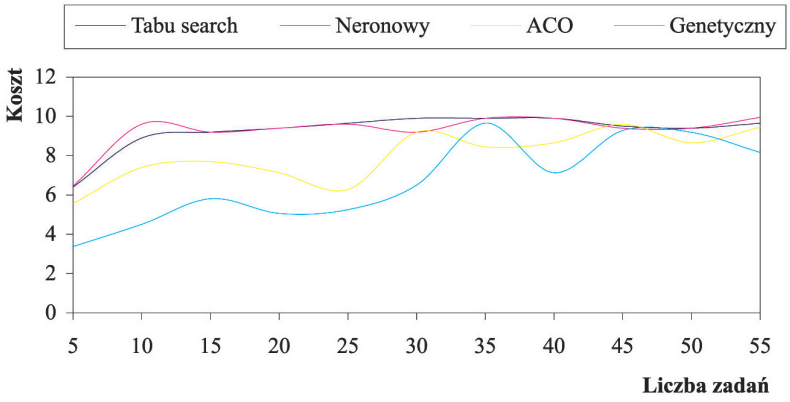
W kolejnym etapie przeprowadzono eksperymenty obliczeniowe porównujące par-syntezę realizowaną przez wszystkie prezentowane metody w celu okreslenia, która z nich realizuje najlepiej syntezę systemową.

Dla badanych problemów przy większej liczbie zadań algorytmy genetyczny i kolonii mrówek generują niższy koszt struktury i krótsze uszeregowania niż inne heurystyki (por. rys. 4 i 5).

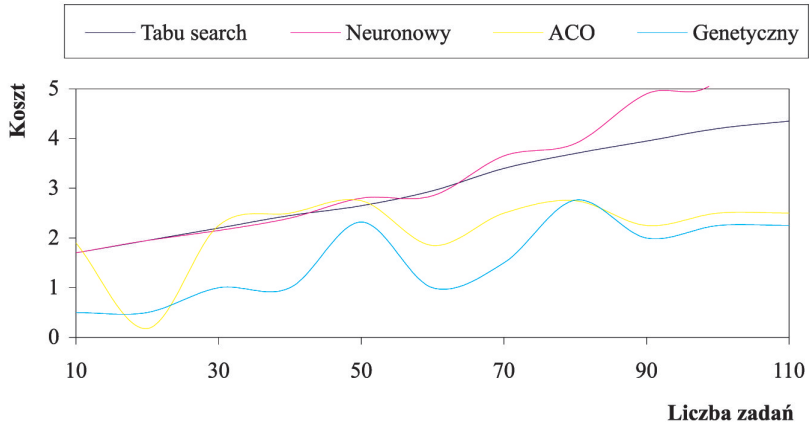
Na podstawie zaprezentowanych zależności można wnioskować, że struktury systemu generowane przez algorytmy genetyczny i kolonii mrówek są tańsze i szybsze. Wraz ze wzrostem liczby zadań algorytmy wykorzystują do tworzenia struktur coraz większą liczbę zasobów (rys. 6), przez co wzrasta koszt, ale jednocześnie maleje czas wykonywania zadań (rys. 7).



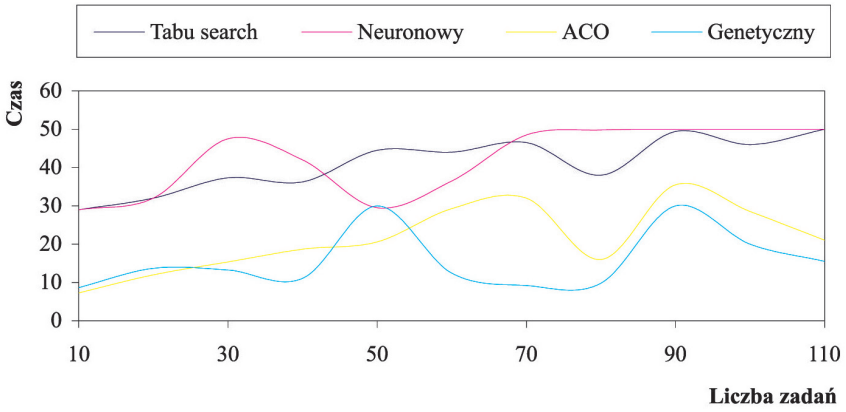
Rys. 4. Minimalizacja czasu wykonania



Rys. 5. Minimalizacja kosztu realizacji systemu



Rys. 6. Minimalizacja kosztu realizacji systemu dla zwiększonej liczby zadań



Rys. 7. Minimalizacja czasu wykonania dla zwiększonej liczby zadań

4. Wnioski

Proces par-syntezy stanowi spójną realizację algorytmu podziału zasobów z algorytmem szeregowania zadań. W związku z tym zbadano, czy jakość uzyskanych rozwiązań jest lepsza od tych, jakie uzyskuje algorytm niekoherentny.

Dla różnych instancji problemów poszczególne algorytmy mogą generować rozwiązania o różnym stopniu sukcesu, a inne mogą przy różnych liczbach zadań osiągać gorsze rezultaty. O przewadze danego algorytmu nad innymi świadczy utrzymywana tendencja zachowywania lepszych rozwiązań, przy zmianie liczby zadań i ograniczeń. Praktycznie, najlepiej analizować rezultaty optymalizacji otrzymane z kilku algorytmów oraz wielu ich przebiegów i dla konkretnych danych wybrać najlepszy.

W problemie minimalizacji kosztu, algorytm koherentny syntezy poprawił pod względem kosztowym rozwiązania otrzymane w syntezie niekoherentnej. Uzyskano również poprawę czasu wykonywania zadań. Z kolei w problemie minimalizacji czasu, algorytm koherentnej syntezy poprawiał czas i to znacznie.

Wyniki uzyskane dla zbiorów zadań nie dały definitywnej odpowiedzi, która z prezentowanych metod optymalizacji daje zawsze lepsze rezultaty. Analizując wyniki algorytmów można wnioskować, że rozwiązania algorytmu ewolucyjnego są najlepsze pod względem czasu, ale są gorsze pod względem kosztu. Pobór mocy systemów wynikowych wygenerowanych przez algorytm ewolucyjny jest niższy od tych, jakie generuje algorytm ACO. Badania dowodzą, że algorytmy ewolucyjny i ACO koherentnej syntezy znajdują rozwiązania, prawie zawsze lepsze niż rozwiązania algorytmów metodą sieci neuronowych i Tabu-Search. Natomiast, na pytanie, który z algorytmów (ewolucyjny czy ACO) daje lepsze rozwiązania, odpowiedź jest zależna od instancji rozpatrywanego problemu.

Par-synteza systemów, jak również synergiczna metodologia projektowania ich struktur i procedur, może mieć praktyczne zastosowanie dla rozwijania narzędzi do automatycznego wspomagania projektowania dla szybkiego prototypowania tego rodzaju systemów.

Literatura

[1] Drabowski M., Wantuch E., *Coherent Concurrent Task Scheduling and Resource Assignment in Dependable Computer Systems Design.* International Journal of Reliability, Quality and Safety Engineering, vol. 13, no. 1. World Scientific Publishing, 15–24, 2006.

[2] Drabowski M., *Parallel synthesisi of computer system*. Monografia nr 225, Wyd. AGH, Kraków 2011.

[3] Dick R.P., Jha N.K., Mogac., *A Multiobjective Genetic Algorithm for Hardware-Software Cosynthesis of Hierarchical Heterogeneous Distributed Embedded Systems*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 17, No. 10, 1998, 920–935.

[4] Drabowski M., *A genetic method for hardware-software par-synthesis*. International Journal of Computer Science and Network Security, vol. 5, No. 22, 2006, 90–96.

[5] Drabowski M., *Tabu Search and Genetic Algorithms in Par-synthesis of Multiprocessors Systems.* [w:] Proceedings of the IASTED International Conference on Artificial Intelligence Applications. Innsbruck, Austria, ACTA Press, Anaheim, USA, 2009, 146–151.

[6] Wang C.J., Tsang E.P.K., *Solving constraint satisfaction problems using neural-networks*. IEEE Second International Conference on Artificial Neural Networks, 1991.

[7] Drabowski M., *Solving Resources Assignment and Tasks Scheduling Problems using Neural Networks*. Artificial Intelligence Studies, vol. 2/9, 2008, 10–18.

[8] Blum C., *Beam-ACO – Hybridizing ant colony optimization with bean search*. An application to open shop schedling, Comput. Oper. Res., 32, 2005, 1565–1591.

[9] Drabowski M., *The ant colony in par-synthesis of computer system*. In Proceedings of the 11th IASTED International Conference on Artificial Intelligence and Soft Computing, Palma de Mallorca, ACTA Press, Anaheim, USA, 244–249, 2007. http://www.actapress.com/Content_of_Proceeding.aspx?proceedingID=456

[10] Michlowicz E., *Nowe wyzwania dla logistyki produkcji*. Gospodarka Materiałowa & Logistyka, R. 62, nr 12/2010, 58–61.