Zeszyty Naukowe WWSI, No 15, Vol. 10, 2016, s. 41-52

Algorytmy konstrukcyjne dla problemu harmonogramowania projektu z ograniczonymi zasobami

Marcin Klimek*

Państwowa Szkoła Wyższa w Białej Podlaskiej, Zakład Informatyki

Abstrakt

W artykule opisany jest problem harmonogramowania projektu z ograniczoną dostępnością zasobami z kryterium minimalizacji czasu trwania projektu. Do rozwiązania zagadnienia opracowane są algorytmy konstrukcyjne, które mogą być przydatne jako rozwiązania inauguracyjne dla procedur lokalnych poszukiwań. Efektywność proponowanych algorytmów przetestowana jest przy użyciu zadań testowych z biblioteki PSPLIB.

Słowa kluczowe **–** algorytmy konstrukcyjne, harmonogramowanie projektu z ograniczoną dostępnością, heurystyka, reguły priorytetowe

1. Wprowadzenie

Problem harmonogramowania projektu z ograniczonymi zasobami RCPSP (ang. *Resource-Constrained Project Scheduling*) z minimalizacją czasu trwania przedsięwzięcia jest często podejmowanym zagadnieniem badawczym. Jest jednym z problemów silnie NP-trudnych [1], dla których większe praktyczne znaczenie ma znalezienie skutecznych algorytmów heurystycznych. Zastosowanie algorytmów dokładnych tj. procedury podziału i ograniczeń (ang. *branch and bound*) ogranicza

* E-mail: marcin_kli@interia.pl.

Zgłoszono do druku 16 października 2016 r.

się jedynie do harmonogramowania projektów z niewielką liczbą czynności – czas poszukiwań rozwiązania optymalnego dla problemów większych jest nieakceptowalnie duży. Szersze zastosowanie dla RCPSP mają algorytmy przybliżone (heurystyczne), o wielomianowej złożoności obliczeniowej, które znajdują harmonogramy w akceptowalnym czasie, także dla problemów z większą liczbą czynności. Do heurystyk opracowanych dla zagadnień NP-trudnych zaliczyć można algorytmy konstrukcyjne oraz algorytmy lokalnych poszukiwań (ang. *LS – Local Search*). Przegląd stosowanych algorytmów i modeli dla RCPSP można znaleźć w pracach przeglądowych [2-4].

Algorytmy konstrukcyjne tworzą harmonogramy na podstawie prostych reguł priorytetowych (algorytmy priorytetowe) lub reguł wstawiania kolejnych czynności (algorytmy wstawień). Ich zaletą jest szybki czas działania. Są stosowane przede wszystkim do tworzenia rozwiązań inauguracyjnych, początkowych, które są „poprawiane" (jakość rozwiązań dla algorytmów konstrukcyjnych jest często nieakceptowalna) przez algorytmy LS tj. symulowane wyżarzanie (ang. *simulated annealing*), algorytmy genetyczne (ang. *genetic algorithms*), poszukiwanie z zakazami (ang. *tabu search*) itd. Efektywność heurystyk dla RCPSP z kryterium minimalizacji czasu trwania projektu jest potwierdzona w badaniach [3-4].

W tym artykule zaproponowane są algorytmy konstrukcyjne, stosujące metodę wstawień, zbliżone do procedur skutecznych dla permutacyjnego systemu przepływowego (ang. *flow shop*). Efektywność algorytmów przetestowana jest dla problemów testowych z biblioteki *PSPLIB* (ang. *Project Scheduling Problem LIBrary*) [5].

2. Sformułowanie problemu

Projekt (przedsięwzięcie) to unikalny zbiór współzależnych czynności, zadań (w artykule terminy zadanie i czynność są używane zamiennie), które należy zrealizować dla osiągnięcia przyjętych celów, przy użyciu określonych zasobów (pracowników, maszyn, materiałów), W pracy rozważany jest klasyczny problem RCPSP, z niepodzielnymi zadaniami (nie można przerywać ich wykonania ang. *nonpreemptive*), o jednym sposobie ich realizacji (nie występują różne warianty realizacji zadań przy użyciu innych typów zasobów ang. *single-mode*).

Projekty przedstawiane są jako graf skierowany $G(V, E)$, w reprezentacji AON (ang. *Activity-On-Node*), w którym V to zbiór węzłów odpowiadający czynnościom, a E to zbiór łuków opisujących relacje kolejnościowe między czynnościami. Celem harmonogramowania jest znalezienie uszeregowania (czasów rozpoczęcia wszystkich zadań) o minimalnym czasie trwania projektu (patrz wzór 1) przy spełnieniu ograniczeń zasobowych (patrz wzór 2) i kolejnościowych (patrz wzór 3).

$$\min F = s_{n+1}, \tag{1}$$

$$\sum_{i \in J(t)} r_{ik} \le a_k, \qquad \forall t = 1,\ldots, s_{n+1}, \forall k = 1,\ldots, K, \tag{2}$$

$$s_i + d_i \le s_j \qquad \forall (i, j) \in E, \tag{3}$$

gdzie:

- n – liczba czynności projektowych, w grafie $G(V, E)$ dodatkowo wstawiane są dwie czynności pozorne 0 i n+1, reprezentujące wierzchołek początkowy i końcowy grafu,
- i – numer (indeks) zadania,
- s_i – czas rozpoczęcia czynności i,
- s_{n+1} – czas rozpoczęcia zadania pozornego n+1, który jest równy czasowi trwania projektu,
- $J(t)$ – zbiór czynności realizowanych w okresie [t-1, t] ,
- K – liczba typów zasobów odnawialnych,
- k – indeks (numer) typu zasobu,
- r_{ik} – zapotrzebowanie czynności i na zasób typu k,
- a_k – dostępność zasobów typu k,
- d_i – czas trwania czynności i.

Czynności realizowane są przy wykorzystaniu odnawialnych zasobów (ang. *renewable resources*), których liczba jest ograniczona i stała w czasie. W trakcie wykonywania zadań, w każdym momencie t, liczba wykorzystanych zasobów nie może przekraczać wielkości dostępnych równych a_k dla każdego z typów zasobu k = 1, …, K (patrz wzór 2).

3. Algorytmy konstrukcyjne

Dla różnych zagadnień optymalizacyjnych, w tym problemów szeregowania zadań, opracowywane są algorytmy konstrukcyjne opierające się na metodzie wstawień. W algorytmach tych, dla zagadnień harmonogramowania, tworzony jest zbiór rozwiązań próbnych powstałych w wyniku wstawienia czynności na różne pozycje w harmonogramie bieżącym, znalezionym podczas wcześniejszych iteracji algorytmu. Spośród tego zbioru rozwiązań próbnych wybierane jest rozwiązanie o najlepszej wartości funkcji celu, które staje się harmonogramem bieżącym w kolejnej iteracji algorytmu. Algorytmy wstawień składają się najczęściej z dwóch etapów:

1) wstępnego, w którym ustalana jest początkowa lista czynności przez użycie wybranego algorytmu wykorzystującego reguły priorytetowe,

2) zasadniczego, w którym generowany jest ciąg *n* permutacji częściowych, rozpoczynając od permutacji 1-elementowej i kończąc na permutacji *n*-elementowej, każda kolejna permutacja częściowa jest tworzona przy uwzględnieniu poprzedniej permutacji i kolejnego zadania z listy początkowej.

Początkowa lista czynności, kolejność pobierania z niej zadań oraz pozycje przypisywane zadaniom w permutacjach częściowych, są specyficzne dla poszczególnych procedur konstrukcyjnych.

Artykuł proponuje nowe algorytmy konstrukcyjne dla RCPSP, opracowane przez autora [6-8], korzystające z koncepcji procedur wykorzystywanych dla innych zagadnień szeregowania tj. permutacyjny system przepływowy *flow shop* [9-11].

Pierwszy z algorytmów, oznaczony **Alg1**, można przedstawić w następujących krokach:

Krok 1:

Utworzenie listy początkowej L złożonej z wszystkich czynności, które mogą być rozpoczęte w chwili $t = 0$, uporządkowanych na podstawie przyjętej reguły priorytetowej **R**.

Krok 2:

Wstawienie pierwszej czynności z listy L na wszystkie możliwe pozycje na aktualnej liście czynności P (przy uwzględnieniu ograniczeń kolejnościowych) i wygenerowanie w każdym przypadku (dla każdej pozycji wstawienia) harmonogramu częściowego (tzn. takiego, w którym nie wszystkie zadania projektowe są uwzględnione przy harmonogramowaniu). Spośród wszystkich możliwych harmonogramów częściowych wybierany jest ten, dla którego wyznaczane jest najlepsze rozwiązanie oceniane przy przyjętym kryterium optymalizacyjnym harmonogramów częściowych.

Krok 3:

Aktualizacja listy L: usunięcie z niej zadania wstawionego w kroku 2 i wstawienie do niej wszystkich następników tego zadania, których wszystkie poprzedniki są już umieszczona na liście P, z zachowaniem porządku zadań na liście L wynikającego z przyjętej reguły priorytetowania **R**.

Kroki 2-3 są wykonywane aż do momentu umieszczenia wszystkich zadań na liście L, z którego generowany jest, przy użyciu SGS (ang. *Schedule Generation Scheme*) [12], harmonogram będący rozwiązaniem problemu RCPSP.

Oznaczenia w opisie algorytmu: P – lista czynności, z której przy użyciu procedury dekodującej SGS, powstaje harmonogram bieżący, L – lista aktualnie dostępnych czynności, które mogą być rozpoczęte tzn. jej poprzednikami są zadania już umieszczone na liście P).

Kolejne opracowane algorytmy konstrukcyjne, oznaczone jako **Alg2** i **Alg3**, działają w zbliżony sposób jak **Alg1**. Różnice są następujące:

- kolejność czynności na liście *L* może być dowolna, nie ma wpływu na przebieg algorytmu,
- w kroku 2 procedury **Alg2** próbkowane jest wstawienie każdego z zadań z listy *L* na ostatnią pozycję na liście *P* i wybierane na tą pozycję jest to zadanie, dla którego generowany jest najlepszy harmonogram częściowy,
- w kroku 2 procedury **Alg3** próbkowane jest wstawienie każdego z zadań z listy *L* na wszystkie możliwe pozycje listy *P* i wybierane jest to zadanie i ta pozycja wstawienia, dla której generowany jest najlepszy harmonogram częściowy.

W procedurach **Alg1, Alg2, Alg3** w kroku 2 wykonywana jest ocena harmonogramów częściowych. Konieczne jest opracowanie takiej funkcji celu, która pozwoli na znalezienie dobrej jakości harmonogramu złożonego z wszystkich czynności. Jako funkcje celu do oceny harmonogramów częściowych proponowane są:

- funkcja **F1**: minimalizacja czasu trwania wszystkich zadań na liście *P*,
- funkcja **F2**: minimalizacja czasu trwania całego projektu, procedura SGS generuje pełne uszeregowanie dla listy czynności złożonej z zadań z aktualnej listy *P* i następnie z pozostałych zadań projektowych w kolejności wynikającej ze stosowanej reguły priorytetowania **R**.

Tabela 1. Reguły priorytetowania zadań

Reguła	Opis reguły
R0	losowe priorytety czynności
R1	minimalny najwcześniejszy czas rozpoczęcia czynności (ang. *ES – Earliest Start*)
R2	minimalny najpóźniejszy czas rozpoczęcia czynności (ang. *LS – Latest Start*) przy uwzględnieniu czasu zakończenia projektu *duedate*
R3	minimalny najpóźniejszy czas zakończenia czynności (ang. *EF – Earliest Finish*) przy uwzględnieniu czasu zakończenia projektu *duedate*
R4	minimalny najwcześniejszy czas zakończenia czynności (ang. *LF – Latest Finish*)
R5	minimalna różnica między najpóźniejszym a najwcześniejszym możliwym czasem rozpoczęcia czynności (ang. *MINSLK - Minimum Slack*)
R6	minimalna różnica między najpóźniejszym a najwcześniejszym możliwym czasem zakończenia czynności
R7	maksymalna liczba wszystkich następników czynności (ang. *MTS – Most Total Successors*)
R8	maksymalna liczba wszystkich bezpośrednich następników czynności (ang. *MITS – Most Immediate Total Successors*)
R9	najkrótszy czas trwania czynności (ang. *SPT – Shortest Processing Time*)
R10	maksymalna suma czasów trwania danej czynności i jej wszystkich następników (ang. *MTSPT – Most Total Successors Processing Time*)

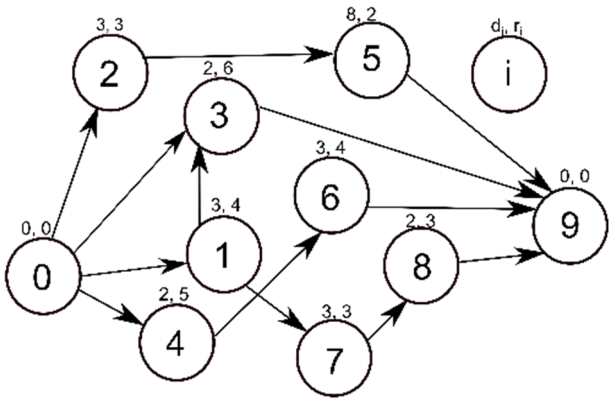
Jako reguły priorytetowe w proponowanych algorytmach wstawień do oceny harmonogramów częściowych (przy wyznaczaniu funkcji **F2**) a także do porządkowania zadań na liście *L* w procedurze **Alg1** stosowane są znane dla RCPSP reguły priorytetowe [6-8,13] przedstawione w tabeli 1.

Jeśli dla danej reguły priorytetowej zadania mają takie same priorytety, na wcześniejszych pozycjach listy czynności umieszczane są zadania oznaczone niższym numerem.

Znalezienie najlepszych reguł priorytetowych, stosowanych do ustalania kolejności zadań na liście *L* oraz porządkowania zadań spoza listy *P*, gdy stosowana jest funkcja **F2** dla oceny uszeregowań częściowych, jest przedmiotem przeprowadzonych w tej pracy eksperymentów.

4. Przykład ilustracyjny

Przykładowy projekt, ilustrujący rozpatrywane zagadnienie, zaprezentowany jest na rysunku 1. Projekt ten składa się z 8 zadań realizowanych przy użyciu jednego typu zasobów o dostępności równej 10.



Rysunek 1. Sieć „czynność na węźle” dla przykładowego projektu

Wierzchołki 0 i 9 przedstawiają początek i koniec $G(V, E)$ – są to zadania pozorne o zerowym czasie trwania i zerowym zapotrzebowania na zasoby. Krawędzie grafu pokazują „technologiczne” zależności kolejnościowe między zadaniami np. czynność 5 może się rozpocząć po zakończeniu czynności 2.

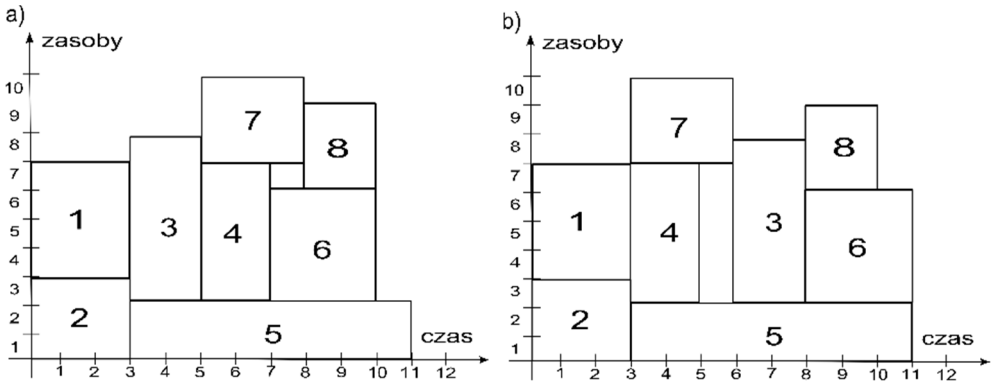
Opracowane algorytmy konstrukcyjne podczas działania korzystają z reguł priorytetowych do ustalania kolejności zadań na liście *L* oraz porządkowania zadań spoza listy *P,* gdy stosowana jest funkcja **F2** dla oceny uszeregowań częściowych.

Do dalszych analiz stosowana jest reguła priorytetowa **R7**, w której zadania są w porządku malejącym na podstawie liczby wszystkich następników zadań (poza zadaniem pozornym nr 9).

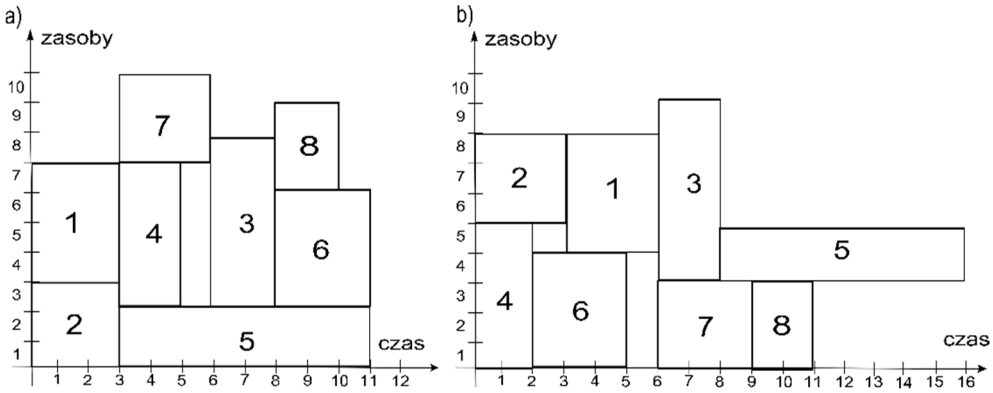
Priorytety poszczególnych czynności są równe:
R7(1) = 3,
R7(2) = 1,
R7(3) = 0,
R7(4) = 1,
R7(5) = 0,
R7(6) = 0,
R7(7) = 1,
R7(8) = 0.

Do generowania rozwiązań, podobnie jak w eksperymentach obliczeniowych, stosowana jest szeregowa procedura dekodująca (ang. *serial SGS*) [12], która na podstawie listy czynności tworzy realizowalny harmonogram (wektor czasów rozpoczęcia zadań): w kolejnych chwilach *t* wyznaczany jest czas rozpoczęcia dla pierwszego nieuszeregowanego zadania z listy czynności, w najwcześniejszym możliwym czasie przy uwzględnieniu ograniczeń kolejnościowych i zasobowych, aż do wyznaczenia czasów rozpoczęcia dla wszystkich czynności.

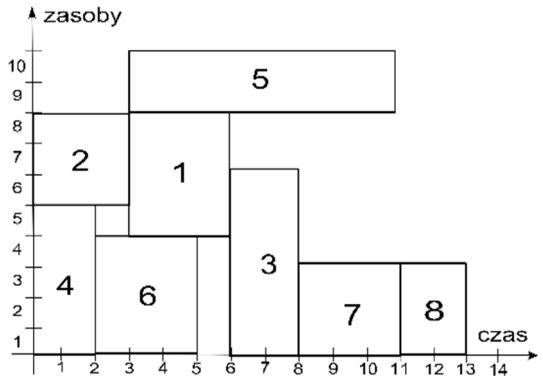
Harmonogramy znalezione przy zastosowaniu opracowanych algorytmów konstrukcyjnych znajdują się na rys. 2, 3 oraz 4.



Rysunek 2. Harmonogramy znalezione przy użyciu **Alg1** przy ocenie harmonogramów częściowych **a)** za pomocą funkcji celu **F2**, b) za pomocą funkcji celu **F1**



Rysunek 3. Harmonogramy znalezione przy użyciu **Alg2** przy ocenie harmonogramów częściowych **a)** za pomocą funkcji celu **F2**, b) za pomocą funkcji celu **F1**



Rysunek 4. Harmonogram znaleziony przy użyciu **Alg3** przy ocenie harmonogramów częściowych za pomocą funkcji celu **F1** i **F2**

Harmonogramy generowane przez poszczególne algorytmy konstrukcyjne różnią się jakością – czasem trwania projektu. Optymalne rozwiązania dla analizowanego, przykładowego projektu, harmonogramy o czasie realizacji równym 11, wygenerowane są przez algorytm **Alg1** i **Alg2** przy funkcji celu **F2** do oceny harmonogramów częściowych.

5. Wyniki eksperymentów

Eksperymenty prowadzono przy użyciu aplikacji zaimplementowanej w języku C# w środowisku Visual Studio.NET, na komputerze z procesorem Intel Pentium 3550M CPU 2.3 GHz, 4 GB RAM, dla 480 instancji testowych ze zbioru J30 (problemy 30-zadaniowe) oraz dla 480 instancji ze zbioru J90 (problemy 90-zadaniowe) z biblioteki *PSPLIB* [5].

Celem eksperymentów była ocena efektywności opracowanych algorytmów konstrukcyjnych dla problemu minimalizacji czasu trwania projektu. Wyniki obliczeń zaprezentowane są w tabeli 2 (dla algorytmu **Alg1**) oraz tabeli 3 (dla procedur **Alg2** i **Alg3**). a średnie czasy działania algorytmów w tabeli 4.

Tabela 2. Wyniki eksperymentów dla algorytmu konstrukcyjnego **Alg1**

	Funkcja celu F1				**Funkcja celu F2**			
	J30		**J90**		**J30**		**J90**	
	a	**b**	**a**	**b**	**a**	**b**	**a**	**b**
R0	11.98%	150	17.14%	143	8.39%	176	12.61%	165
R1	8.04%	164	11.12%	131	5.29%	208	9.29%	171
R2	4.20%	258	4.56%	315	3.12%	274	4.44%	319
R3	3.56%	270	4.54%	308	**2.75%**	**285**	**4.37%**	**313**
R4	8.57%	150	11.93%	125	5.55%	207	9.30%	185
R5	8.08%	208	9.69%	246	5.89%	225	8.42%	256
R6	8.51%	206	10.31%	246	6.74%	213	8.72%	255
R7	4.63%	232	5.53%	269	3.35%	256	5.17%	285
R8	7.47%	187	9.57%	183	5.66%	220	8.36%	208
R9	11.40%	150	16.62%	137	7.82%	184	12.72%	160
R10	4.01%	249	4.96%	287	3.45%	267	4.76%	298

gdzie:

a – średnie procentowe odchylenie względne od najlepszych znanych harmonogramów (optymalnych w przypadku problemów ze zbioru J30);

b – liczba rozwiązań (spośród 480 instancji testowych) o funkcji celu *F* identycznej jak najlepszego znanego harmonogramu (optymalnego dla problemów ze zbioru J30).

Tabela 3. Wyniki eksperymentów dla algorytmów konstrukcyjnych **Alg2** i **Alg3** (oznaczenia jak w tabeli 2)

		Algorytm Alg2				**Algorytm Alg3**			
		J30		**J90**		**J30**		**J90**	
		a	**b**	**a**	**b**	**a**	**b**	**a**	**b**
Funkcja celu F1		13.09%	144	16.54%	128	9.19%	152	13.71%	136
Funkcja celu F2	**R0**	4.79%	223	8.34%	221	7.22%	183	10.03%	186
	R1	4.32%	226	7.37%	219	8.07%	167	10.06%	178
	R2	4.39%	229	7.59%	211	7.38%	175	9.73%	175
	R3	4.47%	230	7.68%	221	7.70%	170	9.85%	177
	R4	4.22%	241	7.57%	221	7.76%	178	10.09%	174
	R5	4.69%	233	8.28%	218	7.68%	184	9.58%	175
	R6	4.88%	224	8.21%	223	7.62%	179	9.93%	194
	R7	4.22%	235	7.47%	218	7.45%	179	10.18%	178
	R8	4.70%	235	7.96%	211	6.85%	180	9.78%	179
	R9	4.91%	228	8.27%	216	7.33%	177	9.92%	188
	R10	4.25%	238	7.63%	227	7.82%	174	9.86%	186

Tabela 4. Średnie czasy działania algorytmów konstrukcyjnych **Alg1**, **Alg2** i **Alg3** (w sek.)

	Funkcja celu F1		**Funkcja celu F2**	
	J30	**J90**	**J30**	**J90**
Alg1	0.004	0.048	0.006	0.182
Alg2	0.004	0.049	0.006	0.196
Alg3	0.007	0.081	0.011	0.389

Najefektywniejszy okazał się algorytm najmniej czasochłonny obliczeniowo: algorytm **Alg1**, przy czym duży wpływ na jego efektywność ma przyjęta reguła priorytetowania zadań na liście *L*. Najlepsze rozwiązania generowane są dla reguł: **R2**, **R3**, **R7**, **R10** przy zastosowaniu funkcji celu **F2** przy ocenie harmonogramów częściowych.

Dla problemów 30-zadaniowych i 90-zadaniowych najlepsze rozwiązania, uzyskano przy zastosowaniu algorytmu **Alg1** z funkcją celu **F2** z regułą priorytetową **R3**. Przy użyciu tego algorytmu rozwiązania znalezione dla projektów ze zbioru J30 są przeciętnie gorsze jedynie o 2.75% od optymalnych (dla problemów 30-zadaniowych znane są rozwiązania optymalne – algorytmy dokładne są w stanie znaleźć uszeregowanie w akceptowalnym czasie) a optymalny harmonogram wygenerowano dla 285 instancji testowych (spośród 480 w zbiorze J30).

Dla projektów złożonych z 90 zadań, harmonogram z czasem realizacji projektu identycznym, jak w najlepszym znanym do tej pory rozwiązaniu, wygenerowano dla 313 instancji testowych (spośród 480 ze zbioru J90) a odchylenie znalezionych rozwiązań od najlepszych znanych wynosi 4.37%.

Wyniki eksperymentów potwierdzają dobrą skuteczność zaproponowanych algorytmów konstrukcyjnych wykorzystujących efektywne reguły priorytetowania zadań.

6. Podsumowanie

W artykule zaproponowano algorytmy konstrukcyjne dla problemu harmonogramowania projektu z ograniczonymi zasobami RCPSP z minimalizacją czasu trwania zadań. Procedury te są oparte na koncepcjach stosowanych dla problemu szeregowania zadań *flow shop*. Działanie procedur sprawdzono dla instancji testowych z biblioteki *PSPLIB*. Eksperymenty wskazały na dobrą efektywność algorytmu **Alg1**, który generuje uszeregowania nieznacznie gorsze od najlepszych znanych (lub optymalnych).

Harmonogramy wygenerowane przez proponowane algorytmy konstrukcyjne mogą być stosowane jako rozwiązania inauguracyjne dla algorytmów lokalnych poszukiwań.

Bibliografia

[1] Błażewicz J., Lenstra J., Kan A.R., *Scheduling subject to resource constraints – classification and complexity*, „Discrete Applied Mathematics" Vol. 5, 1983

[2] Hartmann S., Briskorn D., *A Survey of Variants and Extensions of the Resource-Constrained Project Scheduling Problem*, „European Journal of Operational Research" Vol. 207, No. 1, 2012

[3] Hartmann S., Kolisch R., *Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem*, „European Journal of Operational Research" Vol. 127, 2000

[4] Kolisch R., Padman R., *An integrated survey of deterministic project scheduling*, „OMEGA The International Journal of Management Science" Vol. 29, 2001

[5] Kolisch R., Sprecher A., *PSPLIB – a project scheduling library*, „European Journal of Operational Research" Vol. 96, 1997

[6] Klimek M., *Predyktywno-reaktywne harmonogramowanie produkcji z ograniczoną dostępnością zasobów*, praca doktorska, AGH Kraków, 2010
[7] Klimek M., Łebkowski P. *Algorytmy wstawień dla problemu harmonogramowania projektu z ograniczoną dostępnością zasobów*, w: *Wybrane zagadnienia logistyki stosowanej*, (red.) Bukowski L., Wydawnictwa AGH, Kraków 2009
[8] Klimek M., Łebkowski P., *Algorytmy wstawień dla zagadnienia harmonogramowania projektu ze zdefiniowanymi kamieniami milowym*, w: *Komputerowo zintegrowane zarządzanie*, t. 1, (red.) Knosala R., Oficyna Wydawnicza PTZP, Opole 2010
[9] Nawaz M., Enscore E., Ham I., *A heuristic algorithm for the m machine, n-job flow-shop sequencing problem*, „OMEGA The International Journal of Management Science" Vol. 11, 1983
[10] Woo D.S., Yim H.S., *A heuristic algorithm for mean flowtime objective in flowshop scheduling*, „Computers and Operations Research" Vol. 25, 1998
[11] Nowicki E., Makuchowski M., *Metoda wstawień w klasycznych problemach szeregowania. Cz.* 1. *Problem przepływowy*, w: *Komputerowo zintegrowane zarządzanie*, (red.) Knosala R., t. 2, WNT, Warszawa 2001
[12] Kolisch R., *Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation*, „European Journal of Operational Research" Vol. 90, 1996
[13] Kolisch R., *Efficient priority rules for the resource-constrained project scheduling problem*, „Journal of Operations Management" Vol. 14, 1996

Constructive algorithms for project scheduling with limited resources

Abstract

In this paper resource-constrained project scheduling problem with optimisation criterion of minimising makespan is described. To solve the problem constructive algorithms are developed, which can be useful as an inaugural solution for local search algorithms. The effectiveness of the proposed algorithms is tested using the benchmark instances from the library *PSPLIB*.

Keywords **–** constructive algorithms, resource-constrained project scheduling, heuristic, priority rules