AUTOMATYKA • 2009 • Tom 13 • Zeszyt 2

Marcin Klimek*, Piotr Łebkowski**

Algorytmy odpornej alokacji zasobów dla problemu harmonogramowania projektu z ograniczoną dostępnością zasobów

1. Wprowadzenie

Problem harmonogramowania projektu z ograniczoną dostępnością zasobów *RCPSP* (*Resource-Constrained Project Scheduling Problem*) znajduje liczne zastosowania praktyczne. Dla tego problemu coraz częściej podejmowane jest zagadnienie harmonogramowania odpornego (*robust scheduling*). Harmonogram odporny (proaktywny) definiowany jest jako uporządkowanie zadań, które ze względu na swoje właściwości, jest niepodatne na zakłócenia pojawiające się w trakcie produkcji [1]. Przez odporność rozumiana jest także zdolność harmonogramu do niwelowania skutków nieznacznych wzrostów czasów trwania czynności, które mogą być spowodowane przez niekontrolowane czynniki [2]. Także w tej pracy proponowane są rozwiązania, których celem jest minimalizacja zmian w harmonogramie spowodowanych wahaniami czasów trwania zadań.

Zadaniem odpornego harmonogramowania projektu jest antycypacja zaburzeń produkcyjnych w celu minimalizacji odchylenia planowanego terminu wykonania od rzeczywistego czasu realizacji projektu (podejście *stability of makespan*) oraz/lub w celu minimalizacji zmian (np. w czasach rozpoczęcia zadań czy w przydziale zasobów do czynności) w planowanym harmonogramie w trakcie jego realizacji (podejście *solution robustness*).

Dla problemu *RCPSP* wykonywane są dwa etapy optymalizacyjne, alokacja zasobów (*resource allocation*) i alokacja buforów (*buffer allocation*), które mają na celu stworzenie harmonogramu jak najbardziej odpornego na zakłócenia [3, 4]. Podczas odpornej alokacji zasobów przydzielane są zasoby do realizacji poszczególnych czynności. Natomiast alokacja buforów odbywa się przy ustalonym przydziale zasobów do zadań [5] i polega na wstawianiu buforów czasowych i/lub zasobowych w miejscach najbardziej narażonych na zakłócenia produkcyjne.

* Instytut Informatyki PWSZ Biała Podlaska, doktorant AGH
** Katedra Badań Operacyjnych i Technologii Informacyjnych, Wydział Zarządzania, Akademia Górniczo-Hutnicza w Krakowie

W pracy przedstawiono zagadnienia związane z przydzielaniem zasobów do czynności i wpływem tej alokacji na odporność harmonogramu. W rozdziale 2 opisano model matematyczny problemu, w rozdziale 3 przedstawiono stosowane w badaniach i proponowane przez autorów miary odporności, natomiast w rozdziale 4 zaprezentowano algorytmy alokacji zasobów dla problemu *RCPSP*. Algorytmy przetestowano przy użyciu najczęściej stosowanych w badaniach zadań testowych z biblioteki *PSPLIB* (*Project Scheduling Problem LIBrary*) [6].

2. Sformułowanie problemu

Projekt to zbiór zadań (czynności, operacji). Do reprezentacji projektu stosowana jest sieć „czynność na węźle" *AON* (*Activity On Node*). *AON* to acykliczny, spójny, prosty graf skierowany $G(V, E)$, w którym V to zbiór węzłów odpowiadających zadaniom, a E to zbiór łuków, które opisują relacje kolejnościowe między czynnościami. Zbiór V składa się z n zadań ponumerowanych od 1 do n w porządku topologicznym, tzn. poprzednik ma numer niższy od następnika. Do projektu dołączane są dwa zadania pozorne: 0 i n+1 o zerowych czasach trwania i zerowym zapotrzebowaniu na zasoby, reprezentujące odpowiednio wierzchołek początkowy oraz wierzchołek końcowy grafu $G(V, E)$.

Między czynnościami występują relacje kolejnościowe typu koniec-początek, w których następnik może rozpocząć się bezzwłocznie po zakończeniu poprzednika (*finish-start zero-lag precedence*):

$$s_i + d_i \le s_j, \qquad \forall (i, j) \in E \tag{1}$$

gdzie:

s_i – czas rozpoczęcia zadania i,
d_i – czas wykonywania zadania i.

Zadania realizowane są przez zasoby, których ilość jest ograniczona. W każdym momencie czasu t wykorzystanie zasobów nie może przekraczać wielkości dostępnych [7]:

$$\sum_{i \in A(t)} r_{ik} \le a_k, \qquad \forall t, \forall k \tag{2}$$

gdzie:

a_k – ilość dostępnych zasobów typu k (k = 1, …, K; gdzie K – liczba typów zasobów),
r_{ik} – zapotrzebowanie czynności i na zasób typu k.
$A(t)$ – zbiór zadań wykonywanych w przedziale czasu $[t - 1, t]$.

Zasoby są odnawialne (*renewable*), tzn. ich ilość jest stała (wynosi a_k) niezależnie od obciążeń w poprzednich okresach.

Przy uwzględnieniu podanych powyżej ograniczeń kolejnościowych i zasobowych tworzony jest harmonogram nominalny, czyli szukane są czasy rozpoczęcia zadań s_0, s_1, ..., s_{n+1}.

Najczęstszą funkcją celu jest minimalizacja czasu trwania projektu (*makespan*) równego czasowi rozpoczęcia czynności pozornej końcowej s_{n+1} [7, 8].

Dany harmonogram nominalny może być zrealizowany przy wielu różnych alokacjach zasobów. Ustalenie przydziału zasobów do zadań może chronić przed zmianami w harmonogramie spowodowanymi wydłużeniem czasów trwania zadań. Problem alokacji zasobów dla problemu *RCPSP* jest przedmiotem wielu prac badawczych [3–5, 9–12]. Jest zadaniem silnie *NP*-trudnym, już przy jednym typie zasobów [4].

Do opisu problemu przydziału zasobów wykorzystywana jest sieć przepływu zasobów (*resource flow networks*) [9, 10]. Znajdują się w niej węzły z oryginalnej sieci czynności $G(V, E)$ oraz łuki dodatkowe (zbiór E_R) łączące ze sobą wszystkie pary węzłów (czynności), między którymi występują przepływy zasobów $f(i, j, k)$ (liczby naturalne) dla każdego typu zasobu k od kończącego się zadania i do rozpoczynanego zadania j. W zbiorze E_R znajdują się tylko te łuki, które nie występują w oryginalnej sieci $G(V, E)$. Ograniczenia dla problemu alokacji zasobów można sformułować następująco [3, 4]:

– dla każdego typu zasobów ze zbioru K suma wszystkich zasobów danego typu wychodzących z pozornej czynności początkowej jest równa sumie tych zasobów wchodzących do pozornej czynności końcowej i wynosi a_k:

$$\sum_{j \in V} f(0, j, k) = \sum_{j \in V} f(j, n+1, k) = a_k \qquad \forall k \in K \tag{3}$$

– dla każdego typu zasobów ze zbioru K suma wszystkich zasobów danego typu wchodzących do danego węzła, reprezentującego czynność niepozorną, jest równa sumie tych zasobów wychodzących z tego węzła i wynosi r_{ik}:

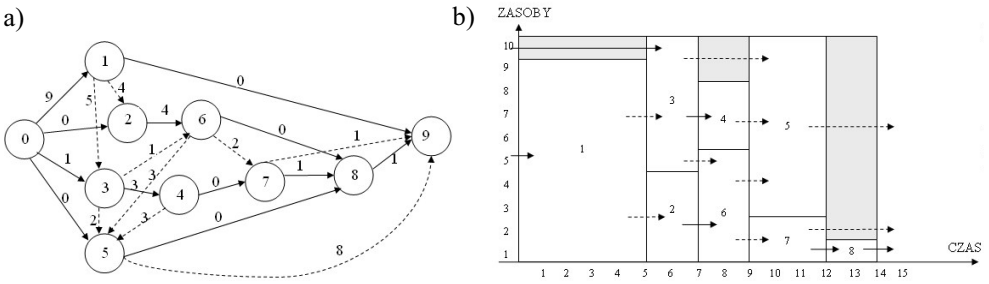
$$\sum_{j \in V} f(i, j, k) = \sum_{j \in V} f(j, i, k) = r_{ik} \qquad \forall i \in V \setminus \{0, n+1\}, \forall k \in K \tag{4}$$

Każdy harmonogram może być zrealizowany przez różne sieci przepływu zasobów, które mogą różnić się pod względem odporności na zakłócenia produkcyjne pojawiające się w trakcie produkcji. W celu redukcji przestrzeni poszukiwań można na początku znaleźć tzw. łuki nieuniknione (*unavoidable arcs*). Zadania i oraz j są połączone łukiem (nieuniknionym) w sieci przepływu zasobów, gdy wymusza to harmonogram nominalny poddawany przydziałowi zasobów. Sytuacja taka ma miejsce, gdy liczba dostępnych zasobów danego typu k (nieuwzględniając zasobów, które zrealizowały czynność i), w momencie rozpoczynania zadania j ($t = s_j$), jest mniejsza niż zapotrzebowanie na zasoby zadania j [3].

Na rysunku 1 przedstawiono harmonogram z przykładową alokacją zasobów dla projektu złożonego z 8 zadań realizowanego przez jeden typ zasobów.

Na rysunkach 1a i 1b oznaczano przerywanymi strzałkami dodatkowe łuki (tworzące zbiór E_R) w sieci przepływu zasobów. Dla sieci przepływu zasobów z rysunku 1a, zbiór E_R składa się z 9 łuków: (1,2), (1,3), (3,5), (3,6), (4,5), (5,9), (6,5), (6,7), (7,9). Spośród nich łuki nieuniknione to: (1,2), (1,3), (3,6), (4,5), (5,9), (6,5), (7,9). Tylko łuki (3,5) i (6,7)

można teoretycznie usunąć ze zbioru E_R przy innej alokacji zasobów. Łuk (3,5) nieznacznie zmniejsza odporność harmonogramu, gdyż przy przepływie zasobów występuje 2-jednostkowy bufor czasowy. Poza tym wyeliminowanie łuku (3,5) prowadzi do powstania łuku dodatkowego (3,7), czyli nie zmniejszy się liczba elementów w zbiorze E_R. Zatem minimalna liczba elementów w zbiorze E_R wynosi 8.



Rys. 1. Sieć przepływu zasobów dla przykładowego projektu złożonego z 8 zadań z zaznaczonymi (przy strzałkach) przepływami zasobów $f(i, j)$ (a); harmonogram z alokacją zasobów (b)

3. Miary odporności alokacji zasobów

Przy odpornej alokacji zasobów należy uwzględnić fakt, że każde wydłużenie czasu realizacji poprzednika powoduje opóźnienie rozpoczęcia następnika (gdy czas rozpoczęcia następnika jest równy czasowi zakończenia poprzednika). Każdy dodatkowy łuk w zbiorze E_R to nowe ograniczenie kolejnościowe, które zmniejsza odporność harmonogramu. W związku z tym problem odpornej alokacji zasobów jest sprowadzany do zagadnienia minimalizacji liczby łuków dodatkowych [3–5, 11, 12]. W tym celu przydziela się czynności, między którymi są zależności kolejnościowe, do realizacji przez te same zasoby [11, 12]. Dąży się także do maksymalizacji sumy przepływów między poszczególnymi zadaniami [3].

Przy ocenie odporności alokacji zasobów analizuje się sieć przepływu zasobów i dąży się do minimalizacji zmian występujących w sieci przepływu zasobów w porównaniu z oryginalną siecią czynności. Jako miara odporności alokacji zasobów stosowana jest elastyczność *flex* (*flexibity*) [8, 9]. Wartość *flex* jest liczona jako stosunek liczby par czynności w sieci przepływu zasobów, między którymi nie ma zależności kolejnościowych do wszystkich możliwych par czynności w projekcie. Elastyczność wskazuje jaka część zadań nie jest powiązana relacjami poprzedzań. Im wyższa wartość *flex*, tym mniejszy stopień zależności między zadaniami i większa odporność harmonogramu. Brak zależności między dwiema czynnościami jest korzystny, gdyż ewentualne opóźnienie zakończenia jednej czynności nie ma wpływu na moment rozpoczęcia drugiej czynności. Problem maksymalizacji wskaźnika *flex* jest równoznaczny z zagadnieniem minimalizacji liczby łuków dodatkowych.

Dla danego uporządkowania nominalnego może istnieć wiele przydziałów zasobów o identycznej wartości *flex*, które są mniej lub bardziej wrażliwe na wydłużenia czasów trwania zadań. Wskaźnik *flex* nie uwzględnia np. podatności na zakłócenia związane z poszczególnymi łukami dodatkowymi. Bardziej zaawansowanym podejściem do mierzenia odporności jest ustalenie wpływu wydłużenia poszczególnych czynności na stabilność produkcji. Funkcją celu alokacji zasobów staje się wtedy funkcja celu harmonogramowania reaktywnego – miara stabilności zrealizowanego harmonogramu. Takie podejście jest stosowane m.in. dla problemu *RCPSP* z minimalizacją ważonego kosztu niestabilności [3–5]. Ważony koszt niestabilności ustalany jest drogą eksperymentalną przy zastosowaniu różnych przebiegów produkcji opracowanych na podstawie wiedzy statystycznej o czasach trwania zadań. W każdym przebiegu losowo generowane są czasy trwania zadań, każdy z rozkładu β o określonych w eksperymencie parametrach.

W niniejszym artykule założono, że nie ma wiedzy statystycznej o zmienności czasów trwania zadań, tzn. wydłużenie planowanego czasu trwania każdego z zadań jest równie prawdopodobne. Czynności są wydłużane z powodów niemożliwych do określenia w fazie planowania, tj. błędy oszacowania czasu trwania, niekorzystne warunki atmosferyczne, awarie itp. Poniżej zaproponowano miary alokacji zasobów, które uwzględniają wpływ wydłużeń poszczególnych zadań na stabilność produkcji, czyli na zakres zmian w zrealizowanym harmonogramie w stosunku do planowanego. Przyjęto, że opóźnienie w rozpoczęciu każdej z czynności ma taki sam wpływ na stabilność produkcji.

Miarą odporności, proponowaną przez autorów, jest funkcja $stab_1$, wyliczana zgodnie z wzorem (5) jako suma opóźnień czasowych w rozpoczęciu wszystkich zadań w stosunku do planu przy założeniu, że każde z zadań jest wydłużone o jedną jednostkę czasową.

$$stab_1 = \sum_{i=1}^{n+1} (s_i^{all} - s_i) \tag{5}$$

gdzie s_i^{all} – przesunięty czas rozpoczęcia zadania i w wyniku wydłużenia każdego z zadań o 1.

W celu ustalenia $stab_1$ tworzony jest zmodyfikowany harmonogram z siecią przepływu zasobów, uwzględniający zmienione czasy trwania zadań (dla każdego zadania $j = 1 \ldots n$ czas realizacji wynosi $d_j + 1$). Prostszą wersją tego miernika jest czas trwania całego projektu dla zmodyfikowanego harmonogramu $stab_2 = s_{n+1}^{all}$.

Następnym proponowanym wskaźnikiem odporności jest funkcja $stab_3$ liczona zgodnie z wzorem (6) jako suma liczby przesuniętych zadań w wyniku opóźnienia każdej z czynności o jedną jednostkę czasową.

$$stab_3 = \sum_{j=1}^{n} \left(\sum_{i=1}^{n+1} (s_i^j - s_i) \right) \tag{6}$$

gdzie s_i^j – czas rozpoczęcia zadania i przy wydłużeniu czynności j o 1.

Liczba opóźnionych zadań (przesuniętych o jedną jednostkę czasową) przy wydłużeniu danego zadania o 1 jest wyznaczana przy założeniu, że pozostałe zadania są realizowane zgodnie z planem. Przesunięcia czasowe wyznaczane są na podstawie analizowanej sieci przepływu zasobów.

Harmonogram o alokacji zasobów z minimalną wartością $stab_1$, $stab_2$ lub $stab_3$ jest odporny na zakłócenia. Przy takiej alokacji ewentualne nieznaczne wydłużenia czynności mają najmniejszy z możliwych wpływów na stabilność realizowanego uszeregowania (opóźnienia innych zadań). Miary $stab_1$, $stab_2$ lub $stab_3$ analizują właściwości harmonogramów, które mogą nie być uwzględnione we wskaźniku *flex*.

4. Algorytmy odpornej alokacji zasobów

Część z algorytmów alokacji zasobów stosuje koncepcje łańcuchów oraz częściowo uporządkowanych harmonogramów *POS* (ang *Partial Order Schedules*). Schemat działania tych algorytmów przedstawiono na rysunku 2.

```
Posortuj wszystkie czynności wg ich czasów rozpoczęcia
Zainicjuj wszystkie łańcuchy poszczególnych zasobów jako puste
for (dla każdego typu zasobów k) do
   for (dla każdej kolejnej czynności j) do
      for 1 to r_jk do //przydziel czynność j do łańcuchów
         m := wybierzŁańcuch (j, k)
         dodaj czynność j na koniec łańcucha m
```

Rys. 2. Schemat działania algorytmu alokacji zasobów POS stosujących koncepcję łańcuchów

Poszczególne algorytmy różnią się procedurą wybierania łańcucha *wybierzŁańcuch*(*j*, *k*). W najprostszym z algorytmów tzw. *BasicChaining* zadania przydzielane są do pierwszych wolnych łańcuchów związanych z kolejnymi zasobami. Powstała sieć przepływu zasobów może być nieodporna na zakłócenia. Do oryginalnej sieci projektów dodawane są (często nadmiarowo) nowe ograniczenia kolejnościowe tzw. punkty synchronizacji, uzależniające rozpoczęcie danej czynności od zakończenia innych czynności (brak jest minimalizacji liczby elementów w zbiorze dodatkowych łuków E_R).

Kolejny z algorytmów tzw. *ISH* (*Iterative Sampling Heuristic*) ma na celu zmniejszenie liczby elementów w zbiorze E_R. Sprowadza się do przydzielania czynności o zapotrzebowaniu na dany zasób ($r_{jk} > 1$), w taki sposób, aby zmaksymalizować liczbę wspólnych łańcuchów (*common chains*), z ostatnimi czynnościami w dostępnych łańcuchach. Dla danej czynności *j* i typu zasobu *k* losowo wybierany jest jeden z dostępnych łańcuchów (łańcuch *m*). Jeśli ($r_{jk} > 1$), zadanie *j* jest przydzielane w pierwszej kolejności do łańcuchów, w których ostatnim elementem jest czynność ostatnia w łańcuchu *m*.

Algorytm *ISH* nie uwzględnia zależności kolejnościowych występujących w oryginalnej sieci projektów. Algorytm ISH^2 przy alokacji zasobów dla danej czynności dodatkowo uwzględnia czynności bezpośrednio poprzedzające daną czynność. W pierwszej kolejności czynność *j* jest przydzielana do łańcuchów, w których ostatnia czynność jest bezpośrednim poprzednikiem czynności *j*. Gdy zapotrzebowanie na zasoby czynności *j* jest większe niż poprzedników tej czynności pozostałe łańcuchy są wybierane zgodnie z algorytmem *ISH*. Zastosowanie algorytmu ISH^2 prowadzi do zmniejszenia liczby elementów w zbiorze dodatkowych łuków E_R, czyli zmniejsza się liczbę punktów synchronizacji.

Autorzy proponują wprowadzić modyfikacje do algorytmów *ISH*, które mają na celu zwiększenie odporności uzyskanego rozmieszczenia zasobów. W pierwszej proponowanej wersji (*ISH-UA*) algorytm działa jak ISH^2, przy czym na początku uruchamiana jest procedura znajdowania łuków nieuniknionych. W pierwszej kolejności czynność *j* jest przydzielana do łańcuchów, w których ostatnia czynność jest bezpośrednim poprzednikiem czynności *j* lub jest połączona łukiem nieuniknionym z czynnością *j*.

W drugiej proponowanej wersji (*ISH-R*) zliczane są liczby wystąpień poszczególnych zadań jako ostatnich czynności na liście dostępnych łańcuchów. Dana czynność *j* jest przydzielana do wspólnych łańcuchów z czynnością o liczbie wystąpień równej zapotrzebowaniu na zasoby czynności *j* (w dalszej kolejności liczbie wystąpień większej o 1 od tego zapotrzebowania itd.). Wybierane są takie łańcuchy, aby liczba punktów synchronizacji była jak najmniejsza przez przydzielanie wspólnych zasobów do zadań powiązanych relacjami kolejnościowymi lub łukami nieuniknionymi.

Dla problemu alokacji zasobów *RCPSP* z funkcją celu wyznaczaną symulacyjnie, którą jest ważony koszt niestabilności stosowane są: algorytm przydziału zasobów tzw. *MABO* (*Myopic Activity-Based Optimization*) oraz procedury całkowitoliczbowe [3]. *MABO* jest to algorytm konstrukcyjny stosujący „krótkowzroczną" optymalizację opartą o analizę wpływu przydziału aktualnie analizowanej czynności *i* do zasobów na wskaźnik stabilności, wyznaczany dla losowo wygenerowanych czasów trwania zadań dla harmonogramu z częściową alokacją zasobów wykonaną do momentu $t = s_i$. Zagadnienie alokacji zasobów można zapisać także jako zerojedynkowy problem programowania całkowitoliczbowego. Dla problemu z funkcją celu minimalizująca ważący koszt niestabilności zdefiniowano trzy procedury optymalizacji całkowitoliczbowej: procedurę *MinEA* (*Minimize Extra Arcs*) – minimalizującą liczbę łuków dodatkowych, procedurę *MaxPF* (*Maximize sum of Pairwise Floats*) – maksymalizującą sumę przepływów zasobów między czynnościami, procedurę *MinED* (*Minimize Estimated Disruptions*) – minimalizującą ważony koszt niestabilności. Czas działania algorytmów programowania całkowitoliczbowego jest znacznie większy niż heurystyk *ISH* czy ISH^2.

5. Wyniki badań eksperymentalnych

Eksperymenty prowadzono przy użyciu 480 instancji testowych z biblioteki *PSPLIB* ze zbioru *J*90, w którym znajdują się projekty złożone z 90 zadań, o jednym sposobie ich

wykonania (*single-mode RCPSP*). Harmonogram nominalny generowano przy użyciu metaheurystyki symulowanego wyżarzania (*Simulated Annealing – SA*). Eksperymentalnie ustalono najlepsze parametry algorytmu i najlepsze techniki przeszukiwania rozwiązań. Przy użyciu *SA* znaleziono uszeregowanie nominalne, czyli czasy rozpoczęcia zadań, stosując jako funkcję celu minimalizację czasu realizacji całego projektu. Następnym etapem jest odporna alokacja zasobów. W tym etapie testowano opisane wcześniej algorytmy *BasicChaining*, *ISH*, *ISH*^{2}, *ISH-UA* oraz *ISH-R*. Jako kryterium oceny przydziału zasobów stosowano miary odporności zaproponowane przez autorów $stab_1$, $stab_2$, $stab_3$ oraz elastyczność *flex*. Wyniki eksperymentów prowadzonych na komputerze klasy Pentium z procesorem 1,7 GHz przy użyciu programu zaimplementowanego w języku C# w środowisku Visual Studio.NET przedstawiono w tabeli 1.

Tabela 1
Wyniki badań eksperymentalnych

Algorytm	*t*	*flex*	$stab_1$	$stab_2$	$stab_3$
BasicChaining	0,008	0,425 (0,167)*	819,4 (279,2)	119,1 (30,4)	1956,6 (761,3)
ISH	0,013	0,432 (0,175)	786,4 (280,8)	118,4 (30,5)	1747,0 (763,0)
ISH^2	0,013	0,447 (0,177)	766,7 (284,9)	117,8 (30,6)	1564,4 (760,2)
ISH-UA	0,029	0,452 (0,174)	752,1 (280,4)	117,5 (30,6)	1469,8 (721,7)
ISH-R	0,029	0,454 (0,173)	750,9 (280,4)	117,5 (30,6)	1460,6 (720,6)

t – czas działania algorytmu w sekundach
* średnia wartość oraz odchylenie standardowe danego miernika (w nawiasach)

Proponowane przez autorów algorytmy *ISH-UA* oraz *ISH-R* tworzą przydziały zasobów bardziej odporne niż uzyskane przy użyciu znanych heurystyk alokacji zasobów. Wskazują na to wszystkie zastosowane mierniki odporności. Wyniki charakteryzują się dużym odchyleniem standardowym, ale wynika to ze zróżnicowania przykładów testowych. Dla większości instancji testowych można zaobserwować przewagę zmodyfikowanych procedur. Przykładowo algorytm *ISH-R* generuje rozwiązania o większej odporności niż najlepszy z algorytmów *ISH*^{2}:

- dla 76,9% (369 z 480) problemów wyższa wartość *flex*,
- dla 94,4% (453 z 480) problemów niższa wartość $stab_1$,
- dla 83,7% (402 z 480) problemów niższa wartość $stab_2$,
- dla 87,7% (421 z 480) problemów niższa wartość $stab_3$.

Dla każdej z miar odporności średnie wartości wskazują, że najlepszy jest algorytm *ISH-R*, następnie *ISH-UA* i w dalszej kolejności procedury *ISH*^{2}, *ISH*. Najsłabszym algorytmem jest *BasicChaining*.

Czas działania *ISH-UA* oraz *ISH-R* jest ponad dwukrotnie większy niż procedur *ISH*, *ISH*^{2}. Jest to związane z koniecznością znalezienia łuków nieuniknionych.

6. Podsumowanie

W artykule zaprezentowano zagadnienie przydziału zasobów dla problemu harmonogramowania projektu z ograniczoną dostępnością zasobów. Skoncentrowano się na określeniu zasad odpornej alokacji zasobów. Zdefiniowano kryteria oceny sieci przepływu zasobów, które mogą być bardziej użyteczne niż stosowane dotąd w badaniach miary odporności. Zaproponowano także modyfikacje znanych heurystyk, które poprawiły odporność uzyskanych rozwiązań.

Przedmiotem dalszych badań będzie m.in. opracowanie skutecznych algorytmów konstrukcyjnych oraz programowania całkowitoliczbowego dla zaproponowanych miar odporności.

Literatura

[1] Jensen M.T., *Improving robustness and flexibility of tardiness and total flow-time job shops using robustness measures*. Applied Soft Computing, 1, 2001, 35–52.

[2] Al-Fawzan M., Haouari M., *A bi-objective problem for robust resource-constrained project scheduling*. International Journal of Production Economics, 96, 2005, 175–187.

[3] Deblaere F., Demeulemeester E.L., Herroelen W.S., Van De Vonder S., *Proactive resource allocation heuristics for robust project scheduling.* Raport badawczy KBI_0608, K.U.Leuven, 2006.

[4] Leus R., *The generation of stable project plans.* Praca doktorska, Katolicki Uniwersytet Lowański, Leuven, Belgia, Katholieke Universiteit Leuven, Belgia, 2003.

[5] Leus R, Herroelen W., *Stability and resource allocation in project planning*. IIE Transactions, 36(7), 2004, 667–682.

[6] Kolisch R., Sprecher A., *PSPLIB – a project scheduling library*. European Journal of Operational Research, 96, 1997, 205–216.

[7] Herroelen W., De Reyck B., Demeulemeester E., *Resource constrained scheduling: a survey of recent developments.* Computers and Operations Research, 25, 1998, 279–302.

[8] Józefowska J., Węglarz J. (red.), *Perspectives in Modern Project Scheduling*. Springer, 2006.

[9] Artigues C., Roubellat F., *A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes.* European Journal of Operational Research, 127, 2000, 294–316.

[10] Artigues C., Michelon P., Reusser S., *Insertion techniques for static and dynamic resource-constrained project scheduling*. European Journal of Operational Research, 149(2), 2003, 249–267.

[11] Policella N., *Scheduling with Uncertainty – A Proactive Approach using Partial Order Schedules*. Praca doktorska, Uniwersytet La Sapienza, Rzym, 2005.

[12] Policella N., Oddi A., Smith S., Cesta A., *Generating robust partial order schedules*. [w:] Proceedings of CP2004, Toronto, Canada, 2004.