EKONOMETRIA ECONOMETRICS 4(38) • 2012

ISSN 1507-3866

Michał Urbaniak
Uniwersytet Ekonomiczny w Poznaniu

ZASTOSOWANIE ALGORYTMU MRÓWKOWEGO DO OPTYMALIZACJI CZASOWO-KOSZTOWEJ PROJEKTÓW INFORMATYCZNYCH

Streszczenie: Harmonogramowanie projektów informatycznych jest problemem specyficznym pośród problemów harmonogramowania projektów z ograniczonymi zasobami, największą bowiem rolę odgrywa w tym przypadku czynnik ludzki i kompetencje wykonawców poszczególnych czynności. Czas trwania czynności projektowych pozostaje w ścisłym związku z kosztem jej wykonania ze względu na zróżnicowanie płacowe występujące wśród specjalistów IT. W pracy zaprezentowano algorytm mrówkowy, za pomocą którego dokonuje się optymalizacji czasowo-kosztowej projektów informatycznych.

Słowa kluczowe: harmonogramowanie projektów, algorytm mrówkowy, projekty informatyczne.

1. Wstęp

Utworzenie pełnego harmonogramu projektu, uwzględniającego nie tylko czasy rozpoczęcia i zakończenia poszczególnych czynności, lecz również zasoby przydzielone do ich wykonania, jest jedną z kluczowych czynności w procesie zarządzania projektami. Nie inaczej jest w przypadku przedsięwzięć, których efektem jest oprogramowanie komputerowe[1]. W sytuacji rynkowej wymagającej od firm dużej konkurencyjności w zakresie terminu i kosztu dostarczenia zamówionego przez klienta oprogramowania, a także w obliczu badań wskazujących na istotną liczbę projektów kończących się przekroczeniem czasu bądź budżetu [*CHAOS Summary…* 2009], harmonogramy optymalizujące wyżej wspomniane czynniki są niezwykle pożądane.

Znany i szeroko opisywany w literaturze problem harmonogramowania projektu z ograniczonymi zasobami (*Resource-Constrained Project Scheduling Problem*, RCPSP) [Alvarez-Valdes, Tamarit 1989; Patterson 1984] nie do końca oddaje naturę projektów informatycznych ze względu na możliwość wykonywania poszczególnych czynności przez różnych wykonawców (pracowników posiadających zróżni-

[1] Autor będzie posługiwał się nazwą „projekty informatyczne", stosując ją w odniesieniu jedynie do projektów wytwarzających oprogramowanie, a nie np. do projektów wdrożeniowych czy dostarczających sprzęt komputerowy.

cowane kompetencje), a co za tym idzie – zmiany czasu i kosztu ich wykonania. Wymagania te uwzględnia problem harmonogramowania projektu o czynnościach wielotrybowych (*Multi-Mode Resource-Constrained Project Scheduling Problem,* MRCPSP). Niniejsza praca jest próbą dostosowania problemu MRCPSP do realiów, w jakich porusza się kierownik projektu informatycznego, i sformułowania oddającego je modelu, a także zastosowanie algorytmu mrówkowego w celu rozwiązania powstałego problemu. Celem pracy jest analiza możliwości wykorzystania algorytmu mrówkowego do optymalizacji czasowej projektów informatycznych.

2. Problem harmonogramowania projektów informatycznych

Problem MRCPSP zakłada, iż czynności projektowe mogą być wykonane w różnych trybach reprezentowanych przez kombinację zasobów przydzielonych do wykonania czynności i odpowiadających im czasów realizacji czynności. Według klasyfikacji dokonanej przez Słowińskiego [1980], w problemach harmonogramowania projektów pod uwagę brane są trzy rodzaje zasobów: odnawialne (np. zasób ludzki, ograniczony tylko w określonej jednostce czasu), nieodnawialne (np. kapitał pieniężny, ograniczony w skali całego projektu) oraz podwójnie ograniczone (w skali zarówno jednostki czasu, jak i całego projektu). Tak określony problem jako swego rodzaju generalizacja RCPSP jest problemem NP-trudnym. Dla problemu MRCPSP zostały wskazane algorytmy zarówno dokładne [Sprecher, Drexl 1997], jak i przybliżone [Alcaraz, Maroto, Ruiz 2003]. W ramach metod heurystycznych najczęściej wykorzystywane w literaturze są algorytmy ewolucyjne, których niewątpliwą zaletą jest łatwość kodowania potencjalnych rozwiązań w ramach MRCPSP, jak również dobra ich „współpraca" z operatorami krzyżowania i mutacji. Obszerną pracę omawiającą problematykę harmonogramowania projektów informatycznych za pomocą algorytmów genetycznych przygotowali Alba, Chicano [2007]. Drugą często wykorzystywaną grupą metod heurystycznych są algorytmy mrówkowe, które w naturalny sposób wpisują się w rozwiązywanie problemów grafowych [Chiang, Huang, Wang 2008; Chen, Zhang 2012].

W problemie harmonogramowania projektu informatycznego (czasami określanego w literaturze jako SPS – *Software Project Scheduling*) najczęściej pomija się nieodnawialne i podwójnie ograniczone zasoby, skupiając się wyłącznie na odnawialnym zasobie ludzkim. Uwagę poświęca się problemowi przydziału pracowników do wykonania poszczególnych czynności ze szczególnym uwzględnieniem optymalizacji wykorzystania kompetencji członków zespołu projektowego [Otero i in. 2009; Barreto, Barrios, Werner 2007]. Warto zwrócić uwagę na zróżnicowane kompetencje, jak również płace wśród specjalistów z sektora IT. Ekspert z dziedziny (np. programowania w języku C++) wykona zapewne czynność w czasie krótszym niż początkujący programista. Otrzyma jednak za swe usługi wyższe wynagrodzenie, co wpłynie niekorzystnie na koszt projektu. Model zaprezentowany w pracy bierze pod uwagę zróżnicowane kompetencje poszczególnych pracowników, a także

proces uczenia się, który powoduje, iż wraz z wykonywaniem coraz większej liczby podobnych do siebie czynności (w naszym przypadku – czynności z tej samej specjalizacji, np. projektowania baz danych) osoba je wykonująca, nabywając doświadczenia, zwiększa swą efektywność. W pracy [Anzanello, Fogliatto 2011] przedstawiono przegląd literatury i sposobów modelowania krzywej uczenia się.

3. Opis modelu decyzyjnego

Projekt informatyczny będzie w proponowanym podejściu modelowany przez zbiór następujących komponentów:

- graf $G = < T, P >$, gdzie T – zbiór węzłów reprezentujących czynności, P – zbiór łuków reprezentujących zależności (ograniczenia technologiczne) między czynnościami,
- zbiór czynności do wykonania $T = \{t_1, t_2, ..., t_N\}$,
- zbiór pracowników $E = \{e_1, e_2, ..., e_M\}$,
- zbiór umiejętności (w ramach których wykonywane są czynności ze zbioru T) $S = \{s_1, s_2, ..., s_K\}$. Inaczej mówiąc, jest to zbiór wszystkich specjalizacji/dziedzin dla czynności projektowych, np. takich jak programowanie w języku C++, testowanie aplikacji, tworzenie dokumentacji analitycznej,
- skalę C, według której oznaczany będzie pomiar kompetencji pracownika e_m w zakresie umiejętności s_k. W pracy przyjmować będziemy, że C = {0, 1, 2, 3, 4, 5}, gdzie 0 oznacza, iż pracownik nie ma w ogóle danej umiejętności, 5 – ma je na najwyższym, eksperckim poziomie.

Uwaga! Zbiór T można również interpretować jako zbiór pakietów zadań (*work packages*), a zbiór E jako zbiór zespołów pracowników, którym przydzielane są elementy ze zbioru T.

W przyjętym podejściu każda czynność ze zbioru T opisywana jest przez trzy wartości:

- liczbę rzeczywistą oznaczającą czas wykonania czynności wskazany przez eksperta (interpretowaną jako czas wykonania czynności przez pracownika o umiejętnościach „średnich" – 3 na przyjętej przez nas skali),
- element ze zbioru S oznaczający, w zakresie jakiej umiejętności wymagane są kompetencje w celu wykonania czynności,
- element ze zbioru C oznaczający minimalny poziom kompetencji wymagany do wykonania czynności (może być interpretowany jako stopień skomplikowania czynności). Jeżeli kompetencje pracownika są niższe – nie może on wykonać danej czynności. W zależności od stopnia, w jakim kompetencje pracownika przewyższają te wymagane przez czynności projektową, harmonogramowany czas wykonania czynności będzie odpowiednio krótszy.

W zakresie sposobu wykonywania czynności przyjęty model opiera się na założeniach, zbliżonych do sformułowanych w [Filho i in. 2007] (z tą jednak różnicą, iż we wspomnianej pracy zakłada się pracę w środowisku wieloprojektowym), takich jak:

– pracownik może wykonywać czynności wymagające różnych ról i umiejętności (jedna osoba może wykonać czynność programistyczną, w innej z kolei odgrywać rolę testera aplikacji),
– pracownik może być przydzielony do realizacji czynności tylko wtedy, gdy ma umiejętności wymagane przez tę czynność do jej prawidłowego wykonania,
– pracownik wykonuje w danym momencie tylko jedną czynność. Czynność ta, raz rozpoczęta, wykonywana jest bez przerwania.

Każdy pracownik będzie w przyjętym modelu charakteryzowany przez następujące wartości:
– liczbę rzeczywistą W oznaczającą stawkę pieniężną płatną pracownikowi za jedną jednostkę czasu (w pracy przyjmować będziemy dzień jako jednostkę czasu),
– K-elementowy ciąg liczb całkowitych będący opisem kompetencji w zakresie umiejętności ze zbioru S (np. programowanie w języku JAVA na poziomie 3, umiejętność testowania aplikacji na poziomie 2, umiejętność projektowania baz danych na poziomie 0).

Problem optymalizacyjny polega na znalezieniu takiego przyporządkowania czynności ze zbioru T – pracownicy ze zbioru E, by przyporządkowanie to nie łamało ograniczeń wynikających z wymagań kompetencyjnych, a czas (koszt) wykonania był jak najniższy. Zakładamy przy tym, iż czas trwania projektu nie powinien przekroczyć maksymalnego, dopuszczalnego przez odbiorcę projektu, czasu jego wykonania.

Przy wprowadzeniu do modelu przepływów pieniężnych (pensji wypłacanych co określoną liczbę jednostek czasu, kosztów nieprodukcyjnych i wpłat od odbiorcy w momentach stanowiących kamienie milowe projektu) problem można zdefiniować jako poszukiwanie optymalnego przyporządkowania $T \to E$, by NPV projektu była jak najwyższa.

W modelu przyjmuje się, iż pracownik, wykonując czynności z tej samej specjalizacji, nabywa umiejętności pozwalające mu zwiększyć wydajność i przyspieszyć wykonanie kolejnych czynności z tej samej specjalizacji. Efekt uczenia się będzie wyznaczany ze wzoru:

$$\tilde{t} = tx^b, \ -1 < b < 0. \tag{1}$$

We wzorze (1) $\tilde{t}$ oznacza czas wykonania czynności z uwzględnieniem efektu uczenia się, t – czas wykonania czynności bez efektu uczenia się, x – liczbę wykonanych czynności z danej specjalizacji, b zaś – parametr, nachylenie krzywej uczenia, charakteryzujący szybkość uczenia się przez pracownika.

Faktyczny czas wykonania czynności przez pracownika obliczany będzie zgodnie ze wzorem:

$$t_{rz} = \tilde{t} \times \left(1 + \frac{3-c}{5}\right), \tag{2}$$

gdzie c oznacza poziom kompetencji niezbędny do wykonania czynności.

4. Algorytmy mrówkowe

Algorytm mrówkowy, zaproponowany pierwotnie przez Dorigo [1992], jest metaheurystyką dobrze sprawdzającą się zarówno w problemach grafowych (np. problem komiwojażera), jak i w wielu problemach optymalizacji dyskretnej (np. problem plecakowy). Pierwsze, wymienione w pracy Dorigo, zastosowanie algorytmu to poszukiwanie najkrótszych ścieżek w grafie oparte na zachowaniu kolonii mrówek poszukujących najkrótszych ścieżek między mrowiskiem a źródłem pożywienia. Wędrujące początkowo w sposób losowy mrówki w przypadku odnalezienia pokarmu powracają do mrowiska, znacząc swą drogę feromonami (oznaczanymi w dalszej części pracy przez τ). Kolejne osobniki nie wędrują w sposób losowy, lecz podążają śladem pozostawionym przez poprzedników. Co prawda feromony wyparowują, obniżając siłę swego przyciągania kolejnych mrówek, jednak krótsze, bardziej pożądane trasy zapewniają większą siłę feromonów, powodując wybór tej właśnie trasy przez następne mrówki. W ten sposób, przez zespołowe działanie kolonii mrówek, odnaleziona zostaje trasa optymalna dostępu do pożywienia. Początkowa idea została rozwinięta i zastosowana zarówno do wielu problemów grafowych, jak i w optymalizacji kombinatorycznej. Powstały również odmiany algorytmu, np. uwzględniające i przechowujące elitę (najlepsze dotychczas znalezione rozwiązania) czy też ograniczające wartości feromonów pozostawionych na szlaku mrówek do przedziału $[\tau_{\min};\tau_{\max}]$.

W pracy, podobnie jak w [Chen i in. 2010], sztuczne mrówki w algorytmie będą poruszały się po skierowanym grafie pełnym, którego wierzchołki oznaczają czynności wraz z przyporządkowanymi im pracownikami (*Mode on Node* – MON). Innymi słowy, w grafie znajdzie się dla każdej czynności tyle wierzchołków, ilu pracowników spełnia dla tej czynności wymagania kompetencyjne. Bez straty ogólności, do grafu G, zdefiniowanego w punkcie 2 pracy, wprowadzone zostają czynności pozorne t_0 oraz t_{N+1} wykonywane tylko w jednym trybie celem ustalenia wierzchołka początkowego i końcowego grafu MON. Czynność t_0 ma tylko jeden następnik – t_1, natomiast czynność t_{N+1} ma tylko jeden poprzednik – t_N. Każda ze sztucznych mrówek musi przejść po grafie MON od wierzchołka początkowego do wierzchołka końcowego tak, by odwiedzić dokładnie jeden wierzchołek grafu dla każdej czynności z T oraz zachować ograniczenia kolejnościowe wynikające z grafu G. W ten sposób problem harmonogramowania zostaje przekształcony w problem przeszukiwania grafu w poszukiwaniu optymalnej ścieżki z wierzchołka początkowego do wierzchołka końcowego. O owej optymalności decydować będzie czas bądź koszt pokonania ścieżki, a więc odpowiadająca jej wartość z powstałego harmonogramu. Warto zauważyć, iż graf MON zawierać może krawędzie nigdy nieużyte przez sztuczne mrówki, ponieważ przejście przez nie złamie ograniczenia kolejnościowe badanego przedsięwzięcia. W celu zabezpieczenia przed powstawaniem rozwiązań niedopuszczalnych w każdym kroku mrówka musi posiadać informację o dostępnych dla niej krawędziach grafu. Realizacja tego wymagania nastąpi przez użycie

szeregowego schematu generowania harmonogramu (*Serial Schedule Generation Scheme* – SSGS). Podczas drogi każdej sztucznej mrówki od źródła do ujścia grafu następuje selekcja następnego wierzchołka spośród trybów dopuszczalnych (tych odpowiadających czynnościom dotąd nieodwiedzonym i jednocześnie niełamiącym ograniczeń kolejnościowych). W tym celu losowana jest liczba rzeczywista q z przedziału [0; 1] i porównywana do liczby q_0 (parametr algorytmu z tego samego przedziału). Jeżeli $q \le q_0$, to wybierany jest ten łuk grafu spośród dopuszczalnych, którego wierzchołek końcowy ma największą wartość spośród $\tau_i \cdot (\eta_i)^{\beta}$ (gdzie τ_i to wartość feromonu dla wierzchołka i, η_i to wartość heurystyczna – pomocnicza dla wierzchołka i, a β to parametr algorytmu sterujący wpływem wartości heurystycznej na wybór wierzchołka). W naszym przypadku wartość heurystyczną wierzchołka w grafie MON zdefiniujemy jako liczbę następników odpowiadającej mu czynności z grafu G, powiększoną o 1, preferujemy zatem czynności, które mają większą liczbę następników. W przypadku, gdy $q > q_0$, uruchamiany jest mechanizm losowy z uwzględnieniem prawdopodobieństw uzyskania odpowiednich wyborów. Prawdopodobieństwo wyboru wierzchołka n_j (spośród wierzchołków dopuszczalnych) obliczane jest zgodnie ze wzorem:

$$p(n_j) = \frac{\tau_j \cdot (\eta_j)^{\beta}}{\sum_{n_i} \tau_i \cdot (\eta_i)^{\beta}}. \tag{3}$$

Po przejściu mrówki przez wybrany wierzchołek następuje aktualizacja wartości feromonu dla tego wierzchołka zgodnie ze wzorem:

$$\tau_i = (1 - \zeta)\tau_i + \zeta, \tag{4}$$

gdzie ζ jest parametrem z przedziału [0; 1]. Efektem takiego działania jest zmniejszenie wartości feromonu, jeżeli ta przekraczała 1, i tym samym pozwolenie kolejnym mrówkom na szerszą eksplorację grafu MON. Po przejściu każdej iteracji algorytmu wyznaczana jest mrówka, która osiągnęła najlepszą wartość funkcji celu (najmniejszy koszt, najkrótszy czas), a feromony na ścieżce, którą podążała, zostają wzmocnione zgodnie ze wzorem:

$$\tau_i = \tau_i + \rho \cdot C^{-1}. \tag{5}$$

W (5) C oznacza najlepszą w danej iteracji wartość funkcji celu, a ρ jest parametrem algorytmu mrówkowego.

5. Algorytm rozwiązania przedstawionego problemu

Ogólny schemat algorytmu przedstawia się następująco:

Krok **0**. Wczytaj dane projektu w postaci piątki < P, T, E, S, C > (dane te zawarte są w pliku wejściowym). Dodaj czynności pozorne t_0 oraz t_{N+1}. Zbuduj graf MON.

Krok 1. Zainicjalizuj parametry algorytmu (m.in. liczbę iteracji *I* oraz liczbę mrówek w każdej z iteracji *J*) oraz wartości feromonów ($\forall_{n_i} \tau_i = 1$). Ponadto wyznacz wartości heurystyczne dla każdego wierzchołka (zgodnie z punktem 3 pracy).

Krok 2. Podstaw $i = 1$.

Krok 3. Podstaw $j = 1$.

Krok 4. Dla mrówki o numerze *j* z iteracji *i* znajdź ścieżkę z wierzchołka początkowego do końcowego w grafie MON z zachowaniem ograniczeń kolejnościowych dla zadań i przy uwzględnieniu wartości feromonów i heurystycznych. Wykorzystaj do tego celu SSGS opisany w punkcie 3 pracy. Dodatkowo zaktualizuj wartości feromonów na ustalonej ścieżce zgodnie ze wzorem (4).

Krok 5. Podstaw $j = j + 1$.

Krok 6. Jeżeli $j \leq J$, to wróć do kroku 4, w przeciwnym przypadku przejdź do kroku 7.

Krok 7. Oblicz wartość funkcji celu harmonogramu projektu wyznaczonego przez wszystkie mrówki z iteracji *i*.

Krok 8. Dokonaj wzmocnienia feromonów na ścieżce, którą kierowała się najlepsza pod względem założonego kryterium mrówka z iteracji *i* zgodnie ze wzorem (5).

Krok 9. Podstaw $i = i + 1$.

Krok 10. Jeżeli $i \leq I$, to wróć do kroku 3, w przeciwnym przypadku przejdź do kroku 11.

Krok 11. Wyświetl najlepsze rozwiązanie z ostatniej iteracji.

Algorytm został zaimplementowany w języku programowania Java i przetestowany na komputerze z procesorem Intel Core 2 Duo T9400 oraz 4 GB pamięci RAM. Wszystkie czasy trwania wykonania algorytmu podane w kolejnych punktach pracy odpowiadają tej konfiguracji sprzętu komputerowego. Zaimplementowany program możemy wywołać w celu optymalizacji czasu lub kosztu wykonania projektu. Dla kierownika projektu pożądanym efektem może być jednak optymalizacja czasowa przy zachowaniu pewnego maksymalnego kosztu wykonania projektu lub optymalizacja kosztowa pod warunkiem, iż projekt wykona się w czasie nie dłuższym niż zakładany. W tym celu możemy przekazać jako parametr programu oznaczenie kryterium, którym się kierujemy, oraz ewentualną maksymalną wartość kryterium drugiego.

W badaniach algorytmu wykorzystywane będą trzy sposoby liczenia kosztów przypadających na jednego pracownika:

I jako liczba jednostek czasu spędzonych nad czynnościami pomnożona przez stawkę jednostkową (podejście wieloprojektowe – w czasie, w którym pracownik nie wykonuje żadnego zadania na rzecz projektu, może wykonywać czynności w ramach innych projektów),

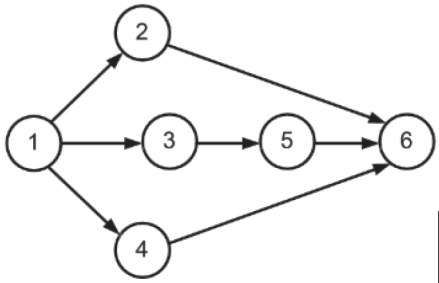
II jako liczba jednostek czasu liczona od momentu rozpoczęcia najwcześniej wykonywanej czynności do zakończenia wykonywanej najpóźniej, pomnożona przez stawkę jednostkową (podejście jednoprojektowe),

III jako iloczyn płacy i czasu (liczby jednostek czasu) trwania projektu.

Koszt wykonania całego projektu liczony będzie jako koszt przypadający na wszystkich pracowników.

6. Przykład liczbowy

W celu zbadania poprawności działania algorytmu posłużono się przykładem liczbowym z rys. 1. W przykładzie projekt składa się z 6 czynności, 3 rodzajów umiejętności oraz 3 pracowników. Wszystkie niezbędne dane podane są w tabelach obok rysunku.



Czynności projektowe			
id	t	s	c
1	30	1	2
2	50	2	1
3	20	2	1
4	20	2	1
5	40	2	1
6	10	2	1

Wykonawcy czynności				
id	W	c1	c2	c3
1	60	3	3	3
2	25	1	1	1
3	75	5	4	1

Rys. 1. Sieć projektu, opis czynności oraz wykonawców czynności

Źródło: opracowanie własne.

Oznaczenia występujące w tabelach są następujące:

t – ekspercki czas wykonania czynności,

s – identyfikator umiejętności niezbędnej do wykonania czynności,

c – poziom umiejętności niezbędny do wykonania czynności,

W – stawka pieniężna pracownika,

c1, c2, c3 – poziomy umiejętności danego pracownika w zakresie kompetencji nr 1, 2 i 3.

Algorytm opisany w punktach 3 i 4 niniejszej pracy został uruchomiony z następującymi parametrami: liczba iteracji – 10, liczba mrówek w każdej z iteracji – 50, $\rho = 0{,}5$, $q_0 = 0{,}5$, $\zeta = 0{,}1$. Czas wykonania algorytmu wyniósł w takim przypadku 203 ms.

Wyniki przedstawione w tab. 1 ukazują najlepsze otrzymane przez program wyniki dla podanego wyżej przykładu liczbowego ze względu na czas i koszt, a także bez uwzględnienia lub z uwzględnieniem efektu uczenia się pracowników.

Tabela 1. Rezultaty otrzymane dla przykładu liczbowego

	Sposób liczenia kosztów	Optymalizacja czasowa harmonogramu	Optymalizacja kosztowa harmonogramu
Bez efektu uczenia się	I	Czas: 76 Koszt: 9250 Pracownicy: 1, 2, 3	Czas: 214 Koszt: 6250 Pracownicy: 2, 3
	II	Czas: 76 Koszt: 9400 Pracownicy: 1, 2, 3	Czas: 214 Koszt: 6250 Pracownicy: 2, 3
	III	Czas: 76 Koszt: 12160 Pracownicy: 1, 2, 3	Czas: 96 Koszt: 9600 Pracownicy: 2, 3
Z efektem uczenia się (b= –0,25)	I	Czas: 65 Koszt: 7720 Pracownicy: 1, 2, 3	Czas: 169 Koszt: 5125 Pracownicy: 2, 3
	II	Czas: 65 Koszt: 8095 Pracownicy: 1, 2, 3	Czas: 169 Koszt: 5125 Pracownicy: 2, 3
	III	Czas: 65 Koszt: 10 400 Pracownicy: 1, 2, 3	Czas: 117 Koszt: 7020 Pracownicy: 1

Źródło: opracowanie własne.

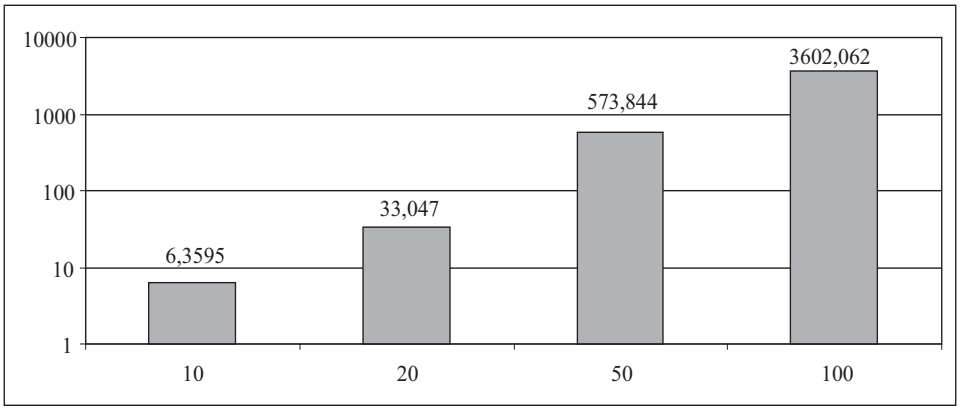
Wyniki potwierdzają intuicyjne podejście – optymalizacja czasowa spowoduje maksymalne wykorzystanie zasobów (a więc zatrudnienie do zespołu projektowego wszystkich dostępnych pracowników), natomiast optymalizacja kosztowa będzie rozsądniej pozyskiwała zasoby (w żadnym z rozpatrywanych przypadków nie zatrudniono trzech osób). Warto zauważyć, że optymalizacja czasowa bez uwzględnienia i z uwzględnieniem efektu uczenia się nie zmienia przypisania osób do czynności, natomiast w przypadku optymalizacji kosztowej, przy III sposobie liczenia kosztów, przyporządkowanie optymalne uległo zmianie.

7. Badanie szybkości algorytmu

Celem sprawdzenia szybkości algorytmu skonstruowany został generator sieci projektowych (program w języku Java, który tworzy pliki xml, przetwarzany następnie przez pierwotny program z zaimplementowanym algorytmem mrówkowym). Uzyskane wyniki przedstawiono na rys. 2 oraz 3.

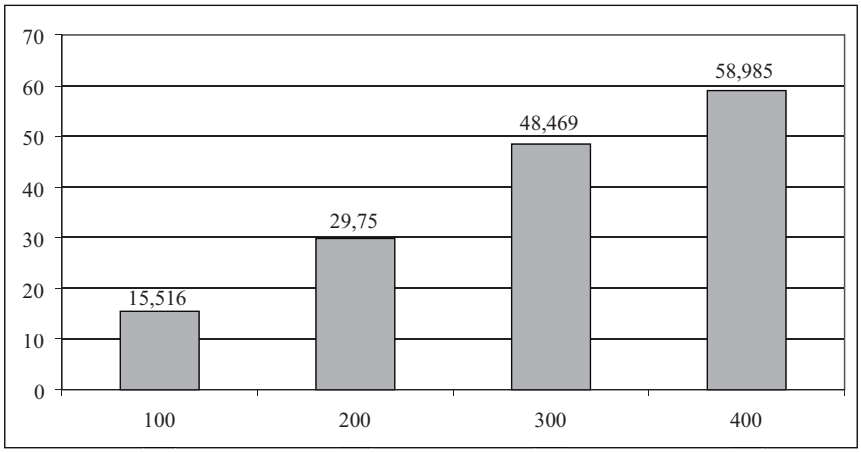
Na rysunku 2 zaprezentowany został średni czas generowania rozwiązań dla problemów o liczbie zadań 10, 20, 50 i 100 (dla każdej wielkości badano 5 losowych instancji problemu) przy liczbie mrówek w każdej iteracji równej 100 oraz liczbie iteracji 100. Wydaje się, iż większa liczba czynności oznacza projekt duży, który powinien być podzielony na fazy rozpatrywane osobno. Warto zwrócić uwagę

na znaczne wydłużenie czasu wykonania algorytmu wraz ze wzrostem liczby czynności. Wynika to z coraz większego skomplikowania grafu MON oraz wyznaczania ścieżki, jaką musi pokonać każda z mrówek.



Rys. 2. Czas wykonania algorytmu w sekundach dla różnych wielkości projektu (liczby czynności)

Źródło: opracowanie własne.

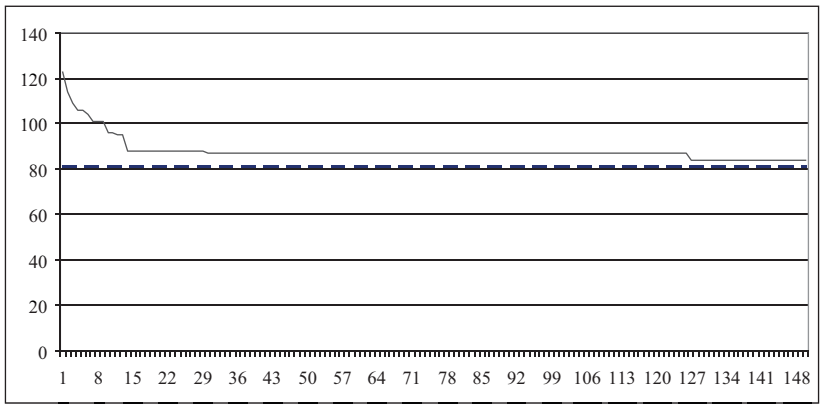


Rys. 3. Czas wykonania algorytmu w sekundach dla projektu o 15 czynnościach i zmiennej liczbie iteracji

Źródło: opracowanie własne.

Na rysunku 3 przedstawiono czas wykonania algorytmu dla problemu o 15 czynnościach, 3 pracownikach oraz zmiennej liczbie iteracji (liczba mrówek w każdej z iteracji – 100). Występuje tu, zgodnie z intuicją, prosta zależność liniowa, a czas wykonania dla danego problemu zależy wyłącznie od liczby mrówek, które mają

przemierzyć graf MON. Należy pamiętać, iż liczba możliwych przyporządkowań (z pominięciem tych, które łamią wymagania kompetencyjne czynności) wynosi 3^{15} = 14 348 907 (ok. 14 milionów). Skala problemu wydaje się niewielka (projekt o 15 czynnościach to zazwyczaj mały projekt), stąd postanowiono sprawdzić, jak długo potrwa przeliczenie zbioru zupełnego rozwiązań. W takim przypadku tworzymy kolejne przyporządkowania, wyznaczając dla każdego z nich harmonogram i koszt tak powstałego projektu. Obliczenia, których celem była optymalizacja czasowa (bez uwzględnienia efektu uczenia się), trwały 170 min (prawie 3 h, 1406 wyznaczonych harmonogramów na sekundę). Na rysunku 4 przedstawiono ewolucję rozwiązania otrzymanego przez algorytm mrówkowy (150 iteracji, 100 mrówek w każdej, czas wykonania: 30 sekund) w stosunku do rozwiązania optymalnego otrzymanego przez przegląd zupełny. Widzimy więc, iż zastosowanie algorytmu powoduje odnalezienie rozwiązania gorszego o 4% od globalnego rozwiązania optymalnego w czasie wielokrotnie krótszym.



Rys. 4. Rozwiązanie optymalne (linia przerywana) a wyniki działania algorytmu mrówkowego podczas poszczególnych iteracji

Źródło: opracowanie własne.

8. Wnioski i dalsze badania

Zaletą zaprezentowanego modelu jest fakt, iż wszystkie informacje o projekcie, niezbędne do rozwiązania zdefiniowanego problemu są łatwe do zdobycia przez kierownika projektu. Dysponuje on zazwyczaj pełną listą czynności wraz z ograniczeniami kolejnościowymi oraz oszacowaniem czasu ich wykonania (szacowania te mogą wykonywać sami pracownicy zespołu projektowego, kierownik projektu lub zewnętrzni eksperci). Nie jest problemem ustalenie, w ramach jakiej dziedziny (specjalizacji) wykonywana jest czynność oraz jakiego poziomu kompetencji wy-

maga. Wyznaczanie kompetencji pracowników wykonywane jest podczas oceny okresowej, wystarczy więc ekstrakcja tych danych z odpowiedniego systemu informatycznego i przekształcenie do formatu wymaganego przez zdefiniowany w pracy model. Zaprezentowany w artykule algorytm rozwiązania problemu jest prosty w implementacji, szybki, a przy tym może być zastosowany do rozwiązywania wielu problemów z zakresu harmonogramowania projektów. Niestety wadą algorytmu, jak również wszystkich algorytmów mrówkowych, są problemy z jego kalibracją i niekiedy duża losowość uzyskiwanych wyników pośrednich.

Przedstawiony w pracy model projektu informatycznego wymaga rozwinięcia, by wierniej oddawał środowisko, w którym porusza się kierownik projektu. W tym celu należałoby wprowadzić do modelu: koszty stałe prowadzenia projektu (koszty zarządu oraz uposażenie kierownika projektu), dostępność pracowników i urlopy, symulacje (np. metodą Monte Carlo) faktycznych czasów trwania projektu (przykład takiej symulacji na podstawie algorytmu mrówkowego przedstawiają np. Chen i in. [2010]), a także reaktywne harmonogramowanie (w przypadku, gdy któraś z czynności została w sposób znaczący opóźniona bądź przyspieszona).

Literatura

Alba E., Chicano J., *Software project management with GAs*, "Information Sciences" 2007, no 177(11).

Alcaraz J., Maroto C., Ruiz R., *Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms*, "Journal of Operational Research Society" 2003, vol. 54.

Alvarez-Valdes, R., Tamarit, J. M., *Heuristic Algorithms for Resource-Constrained Project Scheduling: A Review and an Empirical Analysis. Part I*, [w:] *Advances in Project Scheduling*, R. Slowinski, J. Weglarz (red.), Elsevier Science Publishers B.V., Amsterdam 1989.

Anzanello M.J., Fogliatto F.S., *Learning curve models and applications: Literature review and research directions*, "International Journal of Industrial Ergonomics" 2011, no 41(5).

Barreto A., Barrios M.D.O., Werner C.M.L., *Staffing a software project: A constraint satisfaction and optimization-based approach*, "Computers & Operations Research" 2008, no 35(10).

CHAOS Summary 2009, Raport The Standish Group, Boston 2009.

Chen W.-N. i in., *Optimizing Discounted Cash Flows in Project Scheduling – An Ant Colony Optimization Approach*, IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews 2010.

Chen W.-N., Zhang J., *Ant Colony Optimization for Software Project Scheduling and Staffing with an Event-Based Scheduler*, IEEE Transactions on Software Engineering, 2012.

Chiang C., Huang Y., Wang W., *Ant colony optimization with parameter adaptation for multi-mode resource-constrained project scheduling*, "Journal of Intelligent & Fuzzy Systems" 2008, no 19.

Dorigo M., *Optimization, Learning and Natural Algorithms*, rozprawa doktorska, Politecnico di Milano, Włochy 1992.

Filho A.B. i in., *Staff Scheduling Optimization in Information Technology Projects*, International Conference on Service Systems and Service Management, 2007.

Iredi S., Merkle D., Middendorf M., *Bi-criterion optimization with multi colony ant algorithms, "*Lecture Notes in Computer Science" 2001, vol. 1993.

Otero L.D. i in., *A systematic approach for resource allocation in software projects*, "Computers & Industrial Engineering" 2009, no 56(4).

Patterson J., A *comparison of exact procedures for solving the multiple constrained resource project scheduling problem*, "Management Science" 1984, 30, 7.

Słowiński R., *Two approaches to problems of resource allocation among project activities – a comparative study*, "The Journal of the Operational Research Society" 1980, vol. 31.

Sprecher A., Drexl A., *Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm*, "European Journal of Operational Research" 1998, vol.107.

ANT COLONY SYSTEM APPLICATION FOR TIME-COST OPTIMIZATION OF SOFTWARE PROJECTS

Summary: Scheduling software projects is a specific problem among resource-constrained project scheduling problems because of the huge role of the human factor and competence of individual operations contractors. Duration of activity is closely related to the cost of implementation because of the wage differences that occur among IT professionals. The paper presents an ant colony system used to perform the time-cost optimization of IT projects.

Keywords: scheduling projects, ant colony system, software projects.