ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1996
Seria: AUTOMATYKA z. 117 Nr kol. 1337

Maciej HAPKE, Paweł KOMINEK, Roman SŁOWIŃSKI
Politechnika Poznańska

METAHEURYSTYCZNE PROCEDURY ROZWIĄZYWANIA PROBLEMU PROGRAMOWANIA SIECIOWEGO W WARUNKACH NIEPEWNOŚCI

Streszczenie. W pracy rozpatrywany jest problem programowania sieciowego w warunkach niepewności. Niepewność ta wyrażana jest za pomocą liczb rozmytych. Ze względu na dużą złożoność obliczeniową problemu do jego rozwiązania proponuje się wykorzystanie procedur metaheurystycznych: symulowanej relaksacji, przeszukiwania tabu oraz algorytmu genetycznego. Przedstawiono zarys poszczególnych procedur oraz ich adaptację do rozwiązywania rozpatrywanego problemu. Omówiono eksperyment obliczeniowy, który polegał na wyznaczeniu najlepszych parametrów dla poszczególnych procedur metaheurystycznych oraz na porównaniu wyników działania tych procedur na wybranych przykładach.

METAHEURISTIC PROCEDURES FOR SOLVING PROJECT SCHEDULING PROBLEMS UNDER UNCERTAINTY

Summary. The paper considers a project scheduling problem under uncertainty. The uncertainty is modelled by means of fuzzy numbers. Because of high computational complexity of the problem the use of metaheuristic procedures i.e. simulated annealing, tabu search and genetic algorithm is proposed. An outline of the metaheuristic procedures and their adaptation to the project scheduling problem is presented. The comparison of considered metaheuristic procedures and some relevant conclusions are given.

1. Wprowadzenie

Zastosowania praktyczne programowania sieciowego wymagają tworzenia nowych metod i modeli matematycznych. Jednym z istotnych aspektów, który coraz częściej uwzględnia się w nowych modelach, jest niepewność. Niepewność związana jest z niemożliwością precyzyjnego określenia parametrów czasowych poszczególnych operacji. W takich sytuacjach, gdy brak jest danych statystycznych z przeszłości, najbardziej naturalnym podejściem do modelowania niepewności jest zastosowanie liczb rozmytych. Mimo oczywistej przydatności praktycznej takiego uogólnienia programowania sieciowego dotychczas podjęto niewiele prób jego rozwiązania. Pierwsze wyniki zostały podsumowane w pracach [4,5]. W tej pracy wykorzystane jest nowe podejście oparte na uogólnionej rozmytej procedurze szeregującej, której przedstawienie jest treścią innej pracy [7]. W ogólności problem

rozmytego programowania sieciowego polega na takim rozdziale zasobów do zadań, aby zachowując wszystkie ograniczenia (zasobowe i kolejnościowe) osiągnąć jak najlepszą wartość kryterium optymalizacji. Zaznaczyć tu należy, że wszystkie parametry typu czasowego poszczególnych zadań oraz kryterium maksymalnej długości uszeregowania (ang. makespan) są niepewne i będą w tej pracy wyrażane w kategoriach liczb rozmytych. Szczegółowe sformułowanie problemu zawarte zostało w pracy [7].

Deterministyczna wersja problemu programowania sieciowego należy do grupy problemów NP-trudnych [2], a jego uogólnienie do problemu z niepewnymi parametrami czasowymi nie czyni problemu łatwiejszym. Do jego rozwiązywania można zastosować metody dokładne, specjalizowane heurystyki lub metaheurystyki. Metody dokładne (np. metoda podziału i ograniczeń [11]) dają oczywiście rozwiązania optymalne, ale nie mogą być stosowane do rozwiązywania problemów o realnych rozmiarach ze względu na długi czas obliczeń. Specjalizowane heurystyki (np. heurystyki priorytetowe [1,10]) są zwykle bardzo efektywne i dają niezłe rozwiązania. Dużo lepsze rozwiązania można jednak uzyskać stosując procedury metaheurystyczne (symulowana relaksacja, przeszukiwanie tabu lub algorytmy genetyczne). Choć obliczenia metaheurystyczne trwają zwykle dłużej od tych, które wykorzystują heurystyki specjalizowane, charakteryzują się one dobrą zbieżnością do rozwiązań bliskich rozwiązaniu optymalnemu. Tematem tej pracy jest omówienie eksperymentu obliczeniowego dla wyżej wspomnianych procedur metaheurystycznych.

W następnym rozdziale przedstawiony jest zarys poszczególnych procedur wraz z omówieniem stosowanych parametrów oraz operatorów. Sposób konstrukcji rozwiązania sąsiedniego oraz jego wpływ na jakość otrzymanego rezultatu są treścią rozdziału trzeciego. Następny rozdział omawia eksperyment obliczeniowy polegający na porównaniu działania poszczególnych procedur metaheurystycznych. Podsumowanie pracy zawarte jest w rozdziale piątym.

2. Metaheurystyczne metody optymalizacji kombinatorycznej

Działanie algorytmów metaheurystycznych nawiązuje do takich mechanizmów występujących w przyrodzie, jak osiąganie przez ciało stałe minimalnej energii w procesie schładzania (symulowana relaksacja), naturalna selekcja z zakazem cykli (przeszukiwanie tabu) czy też naturalne selekcje genetyczne (algorytm genetyczny). W rozdziale tym zaprezentowane będą w zarysie wszystkie trzy algorytmy.

2.1. Symulowana relaksacja - zarys metody

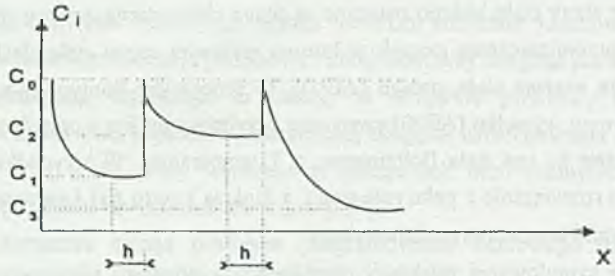
Idea tego algorytmu pochodzi z termodynamiki i nawiązuje do wyżarzania ciał stałych. Charakterystyczną cechą tego procesu jest stopniowe obniżanie temperatury. W procesie

wyżarzania nowe stany ciała stałego osiągane są przez elementarne zmiany stanu bieżącego, tzn. przez małe przemieszczenia cząstek w losowo wybranej części ciała. Jeśli wynikiem tej zmiany jest niższa energia ciała stałego ($\Delta E<0$), to proces jest kontynuowany od nowego stanu. W przeciwnym wypadku ($\Delta E\geq 0$) nowy stan przyjmowany jest z prawdopodobieństwem $\exp(-\Delta E/k_B T)$, gdzie k_B jest stałą Boltzmanna, a T temperaturą. W przypadku optymalizacji kombinatorycznej rozwiązanie x pełni rolę stanu, a funkcja kosztu $f(x)$ i parametr kontrolny C odpowiadają energii i temperaturze ciała stałego.

Procedurę symulowanej relaksacji charakteryzują ponadto: temperatura początkowa (C_0), liczba ruchów na każdym poziomie temperatury (L), warunek stopu, oraz sposób zmiany wartości parametru kontrolnego. Bardzo ważny dla efektywności algorytmu jest sposób określania sąsiedztwa S_x (patrz rozdział 3). Algorytm symulowanej relaksacji można zapisać następująco [9]:

```
procedure SYMULOWANA_RELAKSACJA;
begin
 INICJALIZUJ(x_start,C0,L);
 k:=0;
 x:=x_start;
 repeat
  for l:=1 to L do
  begin
   GENERUJ(x' from Sx);
   if f(x')≤f(x) then
    x:=x'
   else
    if exp(-(f(x')-f(x))/Ck)>random[0,1) then
     x:=x';
  end;
  k:=k+1;
  OBLICZ(Ck);
 until WARUNEK_STOPU;
end;
```

Dodatkowym elementem współdziałającym przy zmianie parametru kontrolnego jest parametr h określający liczbę rozwiązań nie przynoszących polepszenia funkcji kosztu. W przypadku gdy parametr ten osiągnie wartość ekstremalną, następuje zwiększenie wartości parametru kontrolnego (w ten sposób ponowne pogorszenie funkcji kosztu będzie częściej akceptowane). Zmianę parametru kontrolnego C, po uwzględnieniu parametru h przedstawia rys. 1.



Rys. 1. Wykres zmian parametru kontrolnego
Fig. 1. The changes of control parameter

2.2. Przeszukiwanie tabu - zarys metody

Metoda przeszukiwania tabu oparta jest na metodzie iteracyjnego polepszania wyniku funkcji kryterialnej (iterative improvement). Ta druga metoda rozpoczyna obliczenia od rozwiązania początkowego x, a następnie generuje w swoim sąsiedztwie S_x rozwiązanie y. Jeśli $f(y)<f(x)$, to y jest akceptowane jako nowe rozwiązanie. Procedura ta zatrzymuje się, jeśli nie może znaleźć takiego y, że $f(y)<f(x)$. Przeszukiwanie tabu rozpoczyna się właśnie od tego miejsca. Z sąsiedztwa S_x wybierane jest najlepsze rozwiązanie y^* Aby zapobiec powstawaniu w nowych iteracjach rozwiązań, które były niedawno uzyskane, wprowadzono tzw. listę tabu, która zapamiętuje ruchy wykonane w ostatnich $l=|T|$ iteracjach. Parametr l jest w ogólności stałą lub zmienną i oznacza długość listy tabu. Organizacja i rozmiar listy tabu oraz sposób określania sąsiedztwa (patrz rozdział 3) są kluczowymi elementami procedury. Procedurę przeszukiwania tabu charakteryzują ponadto: liczba sąsiadów |V| generowanych w sąsiedztwie S_x oraz warunek stopu. Procedurę przeszukiwania tabu można zapisać następująco:

```
procedure PRZESZUKIWANIE_TABU;
begin
 INICJALIZUJ(xstart,xbest,T);
 x:=xstart;
 repeat
  GENERUJ(V⊆Sx);
  WYBIERZ(x');     {najmniejsza wartość f w zbiorze V}
  UAKTUALNIJ_LISTĘ_TABU(T);
  if f(x')≤f(xbest) then
   xbest:=x';
  x:=x';
  end;
 until WARUNEK_STOPU;
end;
```

2.3. Algorytm genetyczny - zarys metody

Algorytm genetyczny [8] może być opisany jako mechanizm naśladujący ewolucję genetyczną gatunku w kierunku uzyskania wybranych cech. Algorytm ten różni się od dwóch poprzednich głównie tym, że działa nie na pojedynczych rozwiązaniach, lecz na populacji rozwiązań. Rozwiązania te współdziałają ze sobą, mieszają się, tworząc nową generację – „dzieci", które dziedziczą dobre cechy po „rodzicach". Głównymi operatorami odpowiedzialnymi za utworzenie nowej generacji są operatory reprodukcji, krzyżowania oraz mutacji. Literatura dotycząca algorytmu genetycznego bogata jest w terminologię zaczerpniętą z genetyki. Gdy mówimy „chromosom", mamy na myśli zakodowane rozwiązanie. Cecha związana z pojedynczym elementem rozwiązania nazywana jest „genem", wartość „genu" to „allele", a pozycja w rozwiązaniu to inaczej „locus". Ponadto z każdym chromosomem związana jest pewna wartość zwana dopasowaniem, której odpowiada wartość funkcji kryterialnej.

Algorytm genetyczny rozpoczyna działanie od utworzenia populacji N rozwiązań. Rozmiar populacji pozostaje stały przez cały czas działania algorytmu. Generacja $n+1$ tworzona jest z n-tej generacji $X(n)$ w następujący sposób. Z generacji $X(n)$ na podstawie dopasowania wybierane są najlepsze osobniki. Następnie dokonywane są na nich operacje krzyżowania, czego rezultatem jest powstanie potomstwa, które zastąpi złych osobników w bieżącej populacji. Dalej na małej grupce „dzieci" dokonywana jest operacja mutacji. Algorytm genetyczny może być zapisany w następujący sposób:

```
procedure ALGORYTM_GENETYCZNY;
begin
 INICJALIZUJ(X(1));
 for i:=2 to K do
 begin
  WYBIERZ_DOBRYCH_OSOBNIKÓW; {z X(i)}
  KRZYŻOWANIE;
  MUTACJA;
  ZASTĄP_ZŁYCH_OSOBNIKÓW;
 end;
end;
```

gdzie K jest ustaloną liczbą iteracji.

Operatory reprodukcji, krzyżowania oraz mutacji dla problemu rozmytego programowania sieciowego opisane zostaną w rozdziale 4.

3. Konstrukcja rozwiązania sąsiedniego

Zasadniczym warunkiem efektywnego działania algorytmów metaheurystycznych, symulowanej relaksacji i przeszukiwania tabu jest dobre zdefiniowanie sąsiedztwa, które powinno spełniać następujące cechy:

- dla każdego rozwiązania x jego sąsiedztwo N_x powinno zawierać co najmniej jedno rozwiązanie y różne od x oraz nie powinno obejmować zbyt dużej przestrzeni rozwiązań dopuszczalnych,
- każde rozwiązanie $y \in N_x$ powinno niewiele różnić się od x, tak by przejście do nowego rozwiązania nie wymagało konstrukcji nowego rozwiązania "od podstaw",
- osiągalne powinno być każde rozwiązanie należące do przestrzeni rozwiązań dopuszczalnych.

W problemach szeregowania zadań rozwiązanie może być przedstawione na wiele sposobów. Jednym z takich sposobów jest prezentacja rozwiązania za pomocą listy priorytetowej. Lista zawiera uporządkowane zadania, którym procedura szeregująca przydziela zasoby w kolejności malejących priorytetów. Taka lista, choć nie jest rozwiązaniem, w powiązaniu z ustaloną procedurą szeregującą wyznacza w sposób jednoznaczny uszeregowanie (czasy rozpoczęcia i zakończenia poszczególnych zadań). Jedna z prostych definicji sąsiedztwa polega na zamianie miejscami dwóch przypadkowo wybranych zadań na liście priorytetowej. Taka zamiana nie spełnia jednak podstawowych wymagań dobrego sąsiedztwa, gdyż nie zawsze prowadzi do nowego uszeregowania.

W tej pracy proponuje się, aby sąsiedztwo było definiowane na podstawie rozwiązania przedstawianego w postaci wektorów czasów rozpoczęcia i zakończenia poszczególnych zadań [6] według następującego algorytmu:

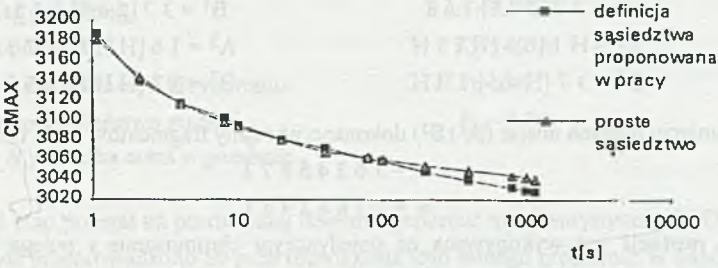
- losuj x z przedziału $[1, n]$
- do nowego rozwiązania kopiuj wszystkie zadania Z_i: $\tilde{t}_i^f <<= \tilde{t}_x^s$
- utwórz zbiór W składający się z zadań Z_j: $\tilde{t}_j^s >>= \tilde{t}_x^s$, spełniających ograniczenia kolejnościowe w chwili $\tilde{t}_x^s$, których $\tilde{a}_i <<= \tilde{t}_x^s$
- losuj Z_y ze zbioru W oraz uszereguj je w chwili $\tilde{t}_x^s$
- uszereguj pozostałe zadania w kolejności ich czasów rozpoczęcia z poprzedniego uszeregowania biorąc Z_x zamiast Z_y,

gdzie: $\tilde{t}_i^s$ – czas rozpoczęcia Z_i,
$\tilde{t}_i^f$ – czas zakończenia Z_i,
$\tilde{a}_i$ – czas gotowości Z_i.

Występujące w powyzszym algorytmie operacje porównywania liczb rozmytych (zaznaczone tyldą) realizuje się według zasady opisanej w [7].

Tak zdefiniowane sąsiedztwo posiada wszystkie cechy dobrego sąsiedztwa wymienione powyżej. Rysunek 2 przedstawia różnice w zbieżności algorytmu symulowanej relaksacji w przypadku zastosowania obu definicji sąsiedztwa. Sposób wyboru rozwiązania sąsiedniego nie

wpływa na jakość obliczeń tylko w początkowej fazie algorytmu, gdzie przeszukiwanie przestrzeni rozwiązań dopuszczalnych ma charakter losowy. Wyraźny jest natomiast wpływ sposobu definicji sąsiedztwa pod koniec obliczeń. Sposób generowania rozwiązań sąsiednich zaproponowany w tej pracy prowadzi do znacznie lepszych wyników minimalizacji funkcji kryterialnej.



Rys. 2. Zależność średniej wartości Cmax od czasu obliczeń dla różnych definicji sąsiedztwa
Fig. 2. Dependency of mean value of Cmax on the computation time for different definition of the neighborhood

4. Operacje reprodukcji, krzyżowania oraz mutacji

W algorytmie genetycznym chromosomy będą odpowiadać listom priorytetowym, a geny zadaniom. Główne operacje odpowiedzialne za utworzenie nowej generacji: reprodukcja, krzyżowanie oraz mutacja, zmieniają kod genetyczny chromosomu tak bardzo, że istnieje tylko bardzo małe prawdopodobieństwo, że dla takiego sposobu kodowania rozwiązania dzieci będą identyczne z rodzicami. Poniżej opiszemy adaptację tych trzech kluczowych operacji do rozważanego tu problemu programowania sieciowego.

Reprodukcja polega na losowym wyborze chromosomów do stworzenia nowej populacji. Im lepsze dopasowanie, tym większe prawdopodobieństwo wyboru chromosomu. Proponuje się, by chromosom $n1$ został wybrany, gdy $\lambda_{n1}>\lambda_{n1}$:

gdzie: $\lambda_{n1} = 1-\frac{\sqrt{\beta}}{10}, \lambda_{n2} = \frac{F(x_i)-F_{min}}{F_{max}-F_{min}}$,

β – wartość losowa z przedziału [1 ; 10^9],

F_{min}, F_{max} – najlepsze i najgorsze dopasowania chromosomu.

Zastosowana tutaj operacja **krzyżowania** LOX (krzyżowanie lokalno-porządkowe [8]) polega na wyszukaniu genów występujących w granicach krzyżowania jednego chromosomu w drugim chromosomie i odwrotnie. Następnie wyszukane geny usuwane są z chromosomów, a puste miejsca przesuwa się tak, by znalazły się w granicach krzyżowania. W następnym kroku następuje wymiana zaznaczonych fragmentów kodu. Dla większej przejrzystości działanie tej operacji zaprezentujemy na przykładzie chromosomu zawierającego osiem genów (lista priorytetowa ośmiu zadań).

Niech dane będą dwa chromosomy A = 4 1 6 8 2 7 3 5 i B = 3 7 2 4 5 1 6 8. Granice krzyżowania wylosowano dla pozycji 3 i 5. W chromosomach A^1 i B^1 wybrano, a w A^2 i B^2 usunięto te geny, które znalazły się w granicach krzyżowania chromosomów B i A odpowiednio.

A = 4 1 [6 8 2] 7 3 5	A^1 = 4 1 [6 8 2] 7 3 5
B = 3 7 [2 4 5] 1 6 8	B^1 = 3 7 [2 4 5] 1 6 8
A^2 = H 1 [6 8 H] 7 3 H	A^3 = 1 6 [H H H] 8 7 3
B^2 = 3 7 [H 4 5] 1 H H	B^3 = 3 7 [H H H] 4 5 1

Po przesunięciu pustych miejsc (A^3 i B^3) dokonano wymiany fragmentów kodu i otrzymano:

A' = 1 6 2 4 5 8 7 3

B' = 3 7 6 8 2 4 5 1

Operacja **mutacji** jest wykonywana na pojedynczym chromosomie i polega na zamianie miejscami dwóch losowo wybranych genów.

5. Eksperyment obliczeniowy: porównanie algorytmów symulowanej relaksacji, przeszukiwania tabu oraz algorytmu genetycznego

Eksperyment obliczeniowy został przeprowadzony w Poznańskim Centrum Superkomputerowo-Sieciowym na superkomputerze SGI Power Challenge (8 procesorów R8000 75 MHz o mocy obliczeniowej 300 MFlops każdy, 512 MB pamięci operacyjnej, 16 GB pamięci dyskowej, pod systemem operacyjnym UNIX w wersji IRIX 6.0).

Doświadczenia przeprowadzono przy zastosowaniu rozmytej procedury szeregującej [7] na problemach składających się z 40, 60, 80 i 100 zadań. Problemy te zostały wygenerowane losowo. Ponieważ nieznane są minimalne wartości maksymalnej długości uszeregowania, dlatego wyniki tu przedstawione odnosić się będą do najlepszych jak dotąd uzyskanych rezultatów optymalizacji. Wynikiem optymalizacji rozmytej jest zawsze rezultat rozmyty. Porównywanie poszczególnych rezultatów rozmytych sprowadza się jednak w naszym przypadku do porównania liczb rzeczywistych odpowiadających środkom ciężkości liczb rozmytych. Dla większej przejrzystości wszystkie rezultaty optymalizacji będą tu zatem wyrażane w takiej właśnie postaci.

Eksperyment przeprowadzono w dwóch etapach. Pierwszy polegał na wyznaczeniu jak najlepszych parametrów dla każdej z badanych metaheurystyk. Dobrano następujące parametry:

symulowana relaksacja:

temperatura początkowa	$C_0=60$
liczba ruchów na każdym poziomie temperatury	$L=0.3*N$
warunek stopu	$e=140*N$
zmiany wartości parametru kontrolnego	$C_i=C_{i-1}*0.95$

przeszukiwanie tabu:

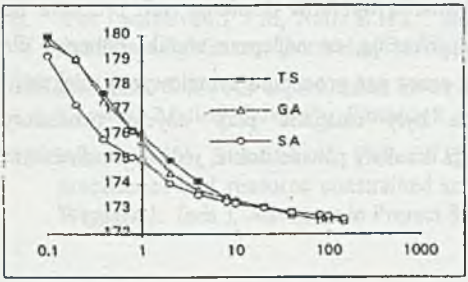
długość listy tabu	$T=0.8*N$
liczba sąsiadów generowanych w sąsiedztwie S_X	$V=0.3*N$
warunek stopu	$e=140*N$

algorytm genetyczny:

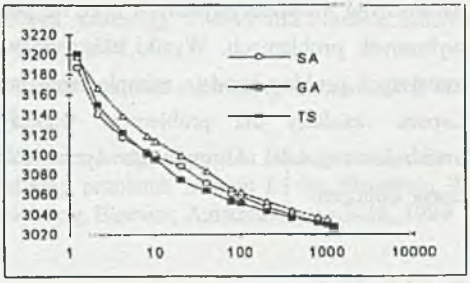
rozmiar pokolenia	$H=0.5*N$
liczba pokoleń	$K=140*N$
prawdopodobieństwo krzyżowania	$P_k=0.6$
prawdopodobieństwo mutacji	$P_m=0.08$

gdzie N jest liczbą zadań w problemie.

Drugi etap polegał na porównaniu działania procedur metaheurystycznych. Dla każdej metaheurystyki przeprowadzono 50 prób rozwiązania tego samego problemu. W każdej próbie przy każdorazowym polepszeniu wartości funkcji kryterialnej zapamiętywano dwie wielkości: wartość funkcji kryterialnej oraz czas, który upłynął od początku obliczeń. Po wykonaniu obliczeń zdyskretyzowano czas obliczeń i dla każdej chwili czasowej obliczono średnią wartość funkcji kryterialnej. Rysunki 3 i 4 przedstawiają średnią wartości maksymalnej długości uszeregowania C_{max} w funkcji czasu obliczeń dla trzech omawianych procedur metaheurystycznych rozwiązujących problemy składające się z 40 i 80 zadań.



Rys. 3. Średnia wartość C_{max} funkcji czasu obliczeń dla problemu "40 zadań"
Fig. 3. Mean value of C_{max} as a function of computation time for the problem of "40 activities"



Rys. 4. Średnia wartość C_{max} funkcji czasu obliczeń dla problemu "80 zadań"
Fig. 4. Mean value of C_{max} as a function of computation time for the problem of "80 activities"

Na podstawie rysunków 3 i 4 można zauważyć, że zaimplementowany przez nas algorytm symulowanej relaksacji osiąga lepsze rezultaty w przypadku problemów mniejszych, o wielkości ok. 40 zadań. Jego przewaga nad pozostałymi jest szczególnie zauważalna w początkowej fazie obliczeń. Najlepsza uzyskana wartość kryterium C_{max}=172.4 została osiągnięta 5-krotnie szybciej dla algorytmu symulowanej relaksacji niż dla przeszukiwania tabu i algorytmu genetycznego.

W przypadku problemów o większych rozmiarach, ok. 80 zadań, można zauważyć, że procedury symulowanej relaksacji i przeszukiwania tabu w pierwszej fazie obliczeń dają wyniki równie dobre. W dalszej części procedura przeszukiwania tabu osiąga jednak lepsze rezultaty. Algorytm genetyczny dla małych problemów daje wyniki gorsze niż algorytm SA, ale lepsze niż TS. .Dla problemów większych wyniki obliczeń algorytmu genetycznego są gorsze od osiągniętych przez algorytmy SA i TS. Algorytm genetyczny osiąga rezultaty procedur SA i TS, ale po dłuższym czasie obliczeń. Powodem tego jest zdolność algorytmu genetycznego do dokładnego przeszukiwania przestrzeni rozwiązań dopuszczalnych [8].

6. Podsumowanie

W pracy zaproponowano rozwiązanie problemu rozmytego programowania sieciowego za pomocą procedur metaheurystycznych: symulowanej relaksacji, przeszukiwania tabu oraz algorytmu genetycznego. Przedstawiono zarys poszczególnych procedur oraz ich adaptację do rozwiązywania rozpatrywanego problemu. W pracy zaprezentowano sposób efektywnego konstruowania rozwiązań sąsiednich wykorzystywany w algorytmach symulowanej relaksacji i przeszukiwania tabu oraz operacje krzyżowania i mutacji charakterystyczne dla algorytmu genetycznego.

Omówiono eksperyment obliczeniowy, który polegał na wyznaczeniu najlepszych parametrów metaheurystycznych oraz na porównaniu wyników działania tych procedur na wybranych problemach. Wyniki eksperymentu pokazują, że najlepsze wyniki obliczeń dla mniejszych problemów daje zaimplementowana przez nas procedura symulowanej relaksacji. Lepsze rezultaty dla problemów większych były osiągane przy użyciu procedury przeszukiwania tabu. Algorytm genetyczny osiąga rezultaty równie dobre, jednak po dłuższym czasie obliczeń.

Podziękowania

Praca ta została wykonana w ramach projektu badawczego KBN Nr 8-S503 016 06. Obliczenia wykonano na zasobach Poznańskiego Centrum Superkomputerowo-Sieciowego. Pierwszy autor pragnie podziękować Fundacji na Rzecz Nauki Polskiej za stypendium, które otrzymuje w roku 1996.

LITERATURA

1. Alvares-Valdés Olaguíbel R., Tamarit Goerlich J.M.: Heuristic algorithms for resource-constrained project scheduling: a review and an empirical analysis, section I.5 in: Słowiński R., Węglarz J.,(eds.), Advances in Project Scheduling, Elsevier, Amsterdam 1989, pp. 113-134.
2. Garey M., Johnson D.: Computers and intractability: A guide to the theory of NP-completeness, Freeman, San Francisco, Calif, 1979.
3. Glover F., Keely J., Laguna M.: Genetic Algorithms and Tabu Search: Hybrids for optimization. Published in University of Colorado, 1992.
4. Hapke M., Słowiński R.: A DSS for resource-constrained project scheduling under uncertainty, Journal of Decision Systems vol. 2 no. 2, 1993, pp.111-128.
5. Hapke M., Jaszkiewicz A., Słowiński R.: Fuzzy Project Scheduling System for Software Development, Fuzzy Sets and Systems 21, 1994, pp. 101-117.
6. Hapke M.: Two-stage fuzzy optimization for project scheduling, Proceedings of the Operational Research of Italy Annual Conference, AIRO'95, Ancona, 20-22 September 1995, pp.295-298.
7. Hapke M.: Programowanie sieciowe w warunkach niepewności, Praca przedłożona do publikacji w Zeszytach Naukowych Politechniki Śląskiej, prezentowana na X KKADPP, Kozubnik, 18-21 września 1996.
8. Goldberg D.: Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.
9. Van Laarhoven P.J.M, Aarts E.H.L.: Simulated Annealing: Theory and Practice, Kluwer Academic Publishers, Dordreht, The Netherlands, 1987.
10. Lawrence S.R.: An experimental investigation of heuristic scheduling techniques, GSIA, Carnegie-Mellon University, Pittsburgh, 1984.
11. Patterson J.H., Słowiński R., Talbot F.B., Węglarz J.: An algorithm for a general class of precedence and resource constrained scheduling problems. section I.1 in: Słowiński R., Węglarz J. (eds.), Advances in Project Scheduling, Elsevier, Amsterdam, pp.3-28, 1989.

Recenzent: Dr inż. Jerzy Mościński

Wpłynęło do Redakcji do 30.06.1996 r.

Abstract

This paper considers a project scheduling problem with uncertain activity time parameters. The uncertainty is modelled by means of fuzzy numbers whose membership functions are defined by six characteristic points. Since the problem is NP-hard it is proposed to solve it using mataheuristic procedures, i.e. simulated annealing, tabu search and genetic algorithm. Although the metaheuristic procedures do not guarantee the optimal solution, they give good suboptimal schedules in relatively short time. In the paper an outline of metaheuristic procedures with an exact description of parameters and operators is presented.

Metaheuristic procedures define only a general scheme of the calculations. This general scheme has to be customized for a given combinatorial problem. The customization consists in defining the way new solutions are obtained. Simulated annealing and tabu search generate new solutions from the neigborhood of a current solution. The paper shows how a good definition of the neighborhood influences the calculation. Genetic algorithms operate on a population of solutions. New population is obtained from the current population after using mutation, crossover and reproduction operators which are described in the paper. The comparison of considered metaheuristic procedures and some relevant conclusions are included in the last part of the paper.