AUTOMATYKA • 2007 • Tom 11 • Zeszyt 1–2

Konrad Wala*

Algorytm tabu w optymalizacji uogólnionego problemu przydziału

1. Wprowadzenie

Uogólniony problem przydziału GAP (*Generalized Assignment Problem*) należy do grupy zagadnień przydziału, z których najbardziej znane to: *Assignment Problem* (szczególny przypadek GAP), *Quadratic Assignment Problem*, *biQuadratic Assignment Problem*, *Min-Max Assignment Problem*, *Balanced Assignment Problem*, *Frequency Assignment Problem*. Wszystkie te problemy mają wspólną cechę: poszukuje się optymalnego przydziału n zadań do m środków, czy to minimalizując funkcję kosztu, czy maksymalizując funkcję zysku. Wielka różnorodność zagadnień przydziału wskazuje, że są to ważne problemy modelujące sytuacje decyzyjne występujące w przemyśle, handlu, transporcie, informatyce i telekomunikacji. Większość ścisłych algorytmów spotykanych w literaturze do rozwiązania GAP stosuje *metodę podziału i ograniczeń* (por.[3]).

W pracy do rozwiązania zagadnienia GAP zastosowano algorytm tabu TS (*Tabu Search*) realizujący proces poszukiwania suboptymalnych rozwiązań w oparciu o metaheurystykę *tabu.* Dobre algorytmy heurystyczne są rozwijane dla konkretnych problemów, ponieważ konieczny jest staranny dobór składowych (np. definicji sąsiedztwa) i parametrów algorytmu w oparciu o dużą liczbę eksperymentów obliczeniowych.

W rozdziale 2 przedstawiono model dyskretny problemu GAP. Rozważany w pracy problem należy do NP-trudnych problemów optymalizacji dyskretnej, stąd celowe jest stosowanie algorytmów przybliżonych zaprezentowanych w rozdziałach 3–5. W rozdziale 6 podano wybrane wyniki badań komputerowych dla instancji z biblioteki OR-library.

2. Model dyskretny problemu GAP

W problemie GAP przydzielany jest zbiór zadań $\Im = \{1, 2, ..., j, ..., n\}$ do zbioru $M = \{1, 2, ..., i, ..., m\}$ środków realizacji zadań (np. maszyn, wykonawców, procesorów), w ten sposób, aby każde zadanie zostało przydzielone do dokładnie jednego środka, zasoby środków nie zostały przekroczone, a całkowity zysk z realizacji wszystkich zadań osiągnął

* Katedra Automatyki, Akademia Górniczo-Hutnicza w Krakowie

wartość maksymalną. Oznaczmy zysk z przydziału zadania j do środka i poprzez c_{ij}, wymagany zasób środka i do wykonania zadania j jako a_{ij} i niech b_i jest dostępną ilością zasobu środka i. Bez straty ogólności można przyjąć, że wartości c_{ij}, a_{ij}, oraz b_i są liczbami całkowitymi.

Niech rozwiązanie problemu jest określone za pomocą funkcji dyskretnej $x : \Im \to M$, która, w dalszym ciągu pracy, będzie formalnie reprezentowana za pomocą następującego ciągu wartości

$$x = (x_1, x_2, ..., x_j, ..., x_n),\ x_j \in M \text{ dla } j \in \Im.$$

Na podstawie funkcji x określane są zbiory zadań Z_i przydzielonych do poszczególnych środków $i \in M : Z_i = \{ j \in \Im : x_j = i\}$. Problem GAP można teraz sformalizować jako następujący problem optymalizacji dyskretnej: znaleźć funkcje x^* maksymalizującą zysk związany z przydziałem zadań do środków

$$v(GAP) = Q(x^*) = \max Q(x) = \max \sum_{i \in M} \sum_{j \in Z_i} c_{ij}$$

i spełniającą następujące ograniczenia wartości zasobów poszczególnych środków

$$\sum_{j \in Z_i} a_{ij} \quad \leq \quad b_i \quad \text{dla} \quad i \in M \tag{1}$$

gdzie v(GAP) jest wartością funkcji celu rozwiązania optymalnego x^*, krótko nazywaną wartością problemu GAP.

W przypadku restrykcyjnych ograniczeń zasobowych trudno jest wyznaczyć metodami przybliżonymi rozwiązanie dopuszczalne. Można wtedy oceniać rozwiązanie przybliżone za pomocą następującej funkcji oceny rozwiązań

$$F(x) = Q(x) - w \cdot V(x) \to \quad \min,$$

gdzie:

$V(x) = \sum_{i=1}^{m} \left| \min\{0, s_i\} \right|$ – funkcja kary za przekroczenie ograniczeń zasobowych środków (1),

$s_i = s_i(Z_i) = b_i - \sum_{j \in Z_i} a_{ij},\ i \in M$ – rezerwa zasobu środka i wyznaczona z nierówności (1),

w – współczynnik wagi, $w \geq 0$.

Zadaniem współczynnika wagi w jest regulacja wpływu wielkości funkcji kary na wartość funkcji oceny rozwiązania $F(x)$. Stosując powyższy wzór, pomijamy ograniczenia związane z wartościami zasobów i poszukujemy rozwiązania w zbiorze wszystkich funkcji dyskretnych $M^{\Im}$.

Przez odpowiednie dobranie współczynnika wagi w, w procesie popraw rozwiązania, zwykle łatwo wyznaczane jest ostatecznie dopuszczalne rozwiązanie przybliżone.

3. Algorytmy konstrukcyjne

Algorytmy konstrukcyjne, oparte na regułach heurystycznych, służą do wyznaczenia rozwiązania początkowego, które następnie jest poprawiane w procesie optymalizacji za pomocą algorytmów popraw. Rozwiązania te niekoniecznie muszą być rozwiązaniami dopuszczalnymi, ponieważ podczas procesu popraw sprowadzane są one do klasy rozwiązań dopuszczalnych (np. poprzez wprowadzenie kary za przekroczenie ograniczeń). W pracy sprawdzano sześć algorytmów szeregowania listowego, każdy o złożoności $O(n\log n)$, gdzie zadaniom przypisano heurystyczny wskaźnik priorytetu $p(j)$, według którego określana jest kolejność przydziału zadań do środków (por. [2, 4]).

Poniżej przedstawione zostały kroki algorytmów konstrukcyjnych szeregowania listowego o numerach $\alpha = 1, 2, ..., 6$.

Krok 1. Oblicz priorytety zadań.
Dla zadań $j \in \Im$ oblicz priorytet:

$$p(j) = \max\left\{ \frac{c_{ij}}{a_{ij}} : i \in M \right\} \text{ – w algorytmach } \alpha = 1, 2, 3,$$

$$p(j) = \max\left\{ c_{ij} : (i \in M) \wedge (a_{ij} < \infty) \right\} \text{ – w algorytmach } \alpha = 4, 5, 6.$$

Krok 2. Sortuj zadania.
Uporządkuj zadania, zgodnie z wartością priorytetu, w ciąg $y = (y(1), y(2), ..., y(n))$ tak, ażeby $p(y(k)) \geq p(y(k+1))$, $k = 1, 2, ..., n-1$.

Krok 3. Ustala warunki początkowe dla procesu konstrukcji rozwiązania.
Dla $i \in M$ podstaw : $s_i := b_i$ i $Z_i := \varnothing$.

Krok 4. Właściwy proces konstrukcji rozwiązania.
Dla $k = 1, 2, ..., n$ wyznacz środek i^{α}, $\alpha = 1, 2, ..., 6$, do którego zostanie przydzielone zadanie $y(k)$:

- $i^1 := \arg\max\left\{ \frac{c_{iy(k)}}{a_{iy(k)}} : i \in M \right\}$ – w algorytmie konstrukcyjnym 1;
  //algorytm ten zwykle znacznie narusza ograniczenia (1);
- $i^2 = i^5 := \arg\max\left\{ s_i - a_{iy(k)} : i \in M \right\}$ – w algorytmach 2 i 5;
  //algorytmy te minimalizują naruszenia ograniczenia (1). Algorytmy konstrukcyjne 2 i 5 różnią się tylko w kroku pierwszym uporządkowaniem zadań według innego wskaźnika priorytetu;
- jeżeli $s_{i^1} - a_{i^1 y(k)} \geq 0$ to w algorytmie konstrukcyjnym 3 podstaw $i^3 := i^1$, w przeciwnym przypadku $i^3 := i^2$;
  //algorytm ten jest wersją mieszaną algorytmów konstrukcyjnych 1 i 2;

- $i^4 := \arg\min\left\{c_{iy(k)} : (i \in M\}) \wedge (a_{iy(j)} < \infty)\right\}$ – w algorytmie konstrukcyjny 4 //algorytm ten w sposób znaczny narusza ograniczenia (1);
- jeżeli $s_{i^4} - a_{i^4 y(k)} \geq 0,$ to w algorytmie konstrukcyjnym 6 podstaw $i^6 := i^4$, w przeciwnym przypadku $i^6 := i^5$;
  //algorytm ten jest wersją mieszaną algorytmów konstrukcyjnych 4 i 5.

Przydziel zadanie $y(k)$ do środka, podstawiając:

$$Z_{i\alpha} := Z_{i\alpha} \cup \{ y(k) \}, \quad s_{i\alpha} := s_{i\alpha} - a_{i\alpha\, y(k)}.$$

4. Sąsiedztwo

Proces poprawy rozwiązania bazowego, rozwiązania poprawianego, x polega na przeszukiwaniu sąsiedztwa $N(x)$ tego rozwiązania. W pracy zastosowano sąsiedztwo $N(x) = N_1(x) \cup N_2(x)$ umożliwiające w czasie O(1) obliczenie wartości funkcji oceny dla każdego $x' \in N(x)$ za pomocą zależności $F(x') = F(x) + w\Delta(x, x')$, gdzie składową $\Delta(x, x')$ można właśnie obliczyć w czasie O(1).

Sąsiedztwo $N_1(x)$ jest podzbiorem rozwiązań x' różniących się od x przydziałem dokładnie jednego zadania:

$$N_1(x) = \{ x' : (x'_j = x_j \text{ dla } \forall j \neq k) \wedge (x'_k \neq x_k) \wedge (k \in \Im)\}$$

i wtedy, w przypadku zmiany przydziału zadania k ze środka x_k na x'_k, obliczamy

$$\Delta(x, x') = s_{x_k}(Z_{x_k} - \{k\}) + s_{x'_k}(Z_{x'_k} \cup \{k\}) - s_{x_k}(Z_{x_k}) - s_{x'_k}(Z_{x'_k}).$$

Sąsiedztwo $N_2(x)$ jest podzbiorem rozwiązań x' otrzymanych z x poprzez zamianę przydziału dwóch zadań do środków:

$$N_2(x) = \{ x' : (x'_j = x_j \text{ dla } \forall j \neq k, l) \wedge (k \neq l) \wedge (x'_k = x_l , x'_l = x_k) \wedge (k, l \in \Im)\},$$

gdzie, w przypadku zamiany przydziału dwóch zadań, obliczamy w sposób podobny zmianę wartości funcji oceny $\Delta(x, x')$.

5. Algorytm tabu

Algorytm popraw rozwiązań tabu TS wykonuje lokalną i powtarzalną modyfikację poprawianego rozwiązania bazowego, z jednoczesnym zapamiętywaniem wykonanych modyfikacji w celu uniknięcia powrotu do sprawdzonych rozwiązań. Przejście w procesie poszukiwania, od rozwiązania bazowego do zmodyfikowanego jest często nazywane ruchem.

Na podstawie pracy [1] zaprojektowano konkretny algorytm tabu TS dla zdefiniowanych struktur rozwiązania x oraz sąsiedztwa $N(x)$, gdzie przyjęto, że poziomem aspiracji jest wartość funkcji celu $Q_{TS} := Q(x_{TS})$ najlepszego rozwiązania x_{TS} znalezionego przez

algorytm TS. Dla ustalenia uwagi i skrócenia opisu algorytmu TS, z jednej strony, a także w celu pokazania niektórych szczegółów zastosowania pamięci krótko- i długo terminowej atrybutów ruchu, z drugiej strony, ograniczono się do prezentacji algorytmu TS eksploatującego tylko sąsiedztwo $N_2(x)$.

Niech dalej rozwiązanie uzyskane w wyniku przestawienia składowej x_i ze składową x_j rozwiązania bazowego x oznaczymy krótko $x(i\ j)$, tj. $x(i\ j) \in N_2(x)$. Przypomnijmy, że rozwiązania sąsiednie w N_2 otrzymujemy zamieniając przydział, na przykład zadań i oraz j, co formalnie polega na przestawieniu składowych x_i oraz x_j w rozwiązaniu x , a zatem atrybutem ruchu jest para zadań (i, j).

Pamięć krótkoterminowa $KLT(i,\ j)$, krótkoterminowa lista tabu, dla atrybutu ruchu (i, j) określa pozostałą liczbę iteracji tabu tego atrybutu. Jeżeli $KLT(i, j) = 0$, to atrybut ruchu (i, j) nie ma warunku tabu, natomiast w przypadku rozwiązania wyznaczonego przez ruch opisany atrybutem (i, j) w algorytmie wykonywane jest podstawienie $KLT(i, j) := T$, gdzie T , $T > 0$, jest parametrem algorytmu ustalającym liczbę iteracji tabu dla atrybutów ruchu. Po wykonaniu każdej iteracji pozostały czas tabu atrybutów jest korygowany za pomocą instrukcji $KLT(i, j) := \max\ \{\ 0, KLT(i, j) - 1\ \}$.

Pamięć długoterminowa DLT , długoterminowa lista tabu, jest pamięcią częstotliwościową stosowaną zarówno do różnicowania procesu przeszukiwania jak i do przerywania długookresowych cyklów przeszukiwania. Do tego podwójnego celu, w procesie sprawdzania jakości rozwiązań niezabronionych sąsiedztwa rozwiązania bazowego, wyznaczana jest wartość funkcji kary $\beta \cdot DLT(i, j)/k$ za stosowanie ruchu (i, j) , gdzie β, $\beta > 0$, jest parametrem algorytmu określającym wagę kary, a k jest numerem iteracji procesu przeszukiwania.

ALGORYTM TS

Krok 1. Początek
Wyznacz rozwiązanie początkowe x_{pocz} za pomocą wybranego algorytmu konstrukcyjnego i podstaw: $x := x_{pocz}$, $x_{TS} := x_{pocz}$, $Q_{TS} := Q(x_{pocz})$, $ch := false$; // ch – zmienna pomocnicza sygnalizująca zastosowanie kryterium aspiracji.

Krok 2. Dla k = 1 do K wykonaj: // *K – zadana liczba iteracji* (warunek stopu):

$$x(i^*\ j^*) = \arg\max\left\{Q\left(x(i\ j)\right) + \frac{\alpha}{k} DLT(i, j) : KLT(i, j) = 0\right\};$$

$$x(i'\ j') = \arg\max\left\{Q\left(x(i\ j)\right) : KLT(i, j) > 0\right\};$$

i podstaw: $x := x(i^*\ j^*)$;

Jeżeli $Q(x) > Q_{TS}$, to $x_{TS} := x$, $Q_{TS} := Q(x)$;
// próba zastosowania kryterium aspiracji

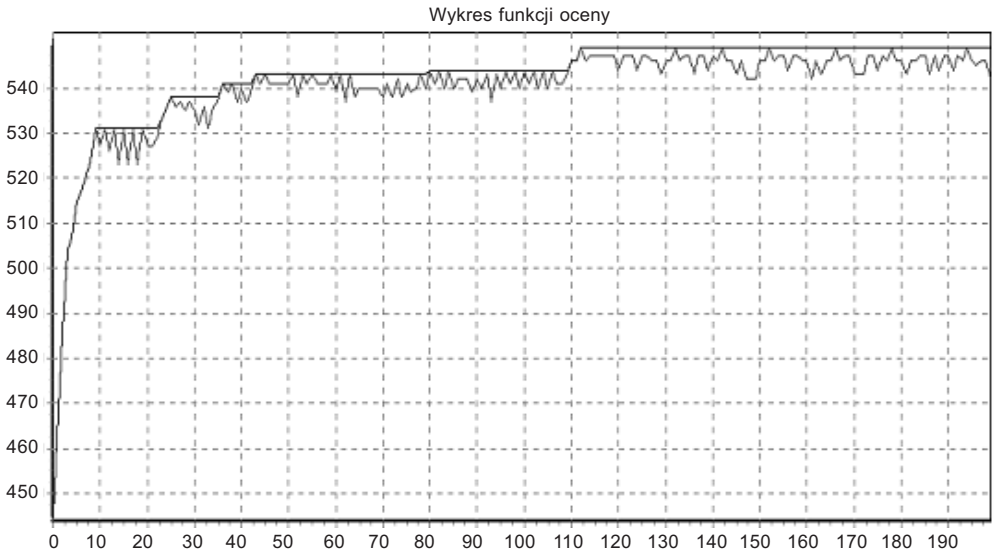
Jeżeli $Q(x(i'\ j')) > Q_{TS}$, to $x := x(i'\ j')$, $x_{TS} := x(i', j')$, $Q_{TS} := Q(x(i', j'))$, $ch := true$;
// korekta pamięci

Dla każdego (i, j) podstaw: $KLT(i, j) := \max\{0, KLT(i, j) - 1\}$;
jeżeli $ch = false$, to $KLT(i^*, j^*) := T$, inaczej $KLT(i', j') := T$;

jeżeli $ch = false$, to $DLT(i^*, j^*) := DLT(i^*, j^*) + 1$, inaczej $DLT(i', j') += 1$, $ch := false$.

6. Wybrane wyniki badań numerycznych i uwagi końcowe

W tabeli 1 przedstawiono wybrane wyniki optymalizacji wykonanych dla instancji z biblioteki OR-library [5]. Przed właściwymi obliczeniami optymalizacyjnymi wykonano znaczną liczbę eksperymentów dla poszczególnych rozmiarów (n, m) instancji w celu dobrania wartości parametrów algorytmu TS oraz wybrania skuteczniejszego algorytmu konstrukcyjnego, przy czym najważniejszy z parametrów, warunek stopu, był ustalany na podstawie obserwacji „stagnacji" procesu optymalizacji.



Rys. 1. Przykład przebiegu procesu optymalizacji

Na rysunku 1 przedstawiono przykład procesu optymalizacji, gdzie zaobserwowano stagnację procesu po 120 iteracjach, można więc zaproponowac: $120 \le K \le 180$. Najlepsze rezultaty w optymalizacji otrzymywano stosując algorytm konstrukcyjny $\alpha = 6$, co raczej nie jest niespodzianką; algorytm ten jest złożeniem algorytmu zachłannego ($\alpha = 4$) i algorytmu z regułą oszczędzania zasobów ($\alpha = 5$).

Wyniki optymalizacji zamieszczone w tabeli 1 otrzymano właśnie, wyznaczając rozwiązania początkowe dla algorytmu TS za pomocą algorytmu konstrukcyjnego $\alpha = 6$, gdzie pierwsza kolumna tablicy zawiera nazwy plików danych charakteryzujących instancje, druga kolumna wartości górnego ograniczenia funkcji celu UB, trzecia kolumna wartości optymalne funkcji celu v(GAP) oraz czwarta wartości funkcji Q_{TS} wyznaczone przez algorytm TS.

Tabela 1
Wyniki badań komputerowych algorytmu TS

Nazwa pliku	*UB*	*v*(GAP)	Q_{TS}
gap5c1_8x24C.dat	574	563	561
gap5c2_8x24C.dat	578	558	556
gap5c3_8x24C.dat	582	564	563
gap5c4_8x24C.dat	586	568	567
gap5c5_8x24C.dat	591	559	559
gap6c1_8x32C.dat	780	761	758
gap6c2_8x32C.dat	777	759	756
gap6c3_8x32C.dat	775	758	757
gap6c4_8x32C.dat	779	752	752
gap6c5_8x32C.dat	771	747	746
gap7c1_8x40C.dat	967	942	939
gap7c2_8x40C.dat	964	949	947
gap7c3_8x40C.dat	979	968	968
gap7c4_8x40C.dat	971	945	940
gap7c5_8x40C.dat	973	951	950
gap8c1_8x48C.dat	1161	1133	1129
gap8c2_8x48C.dat	1166	1134	1132
gap8c3_8x48C.dat	1165	1141	1138
gap8c4_8x48C.dat	1154	1117	1111
gap8c5_8x48C.dat	1159	1127	1123
gap9c1_10x30C.dat	733	709	709
gap9c2_10x30C.dat	740	717	712
gap9c3_10x30C.dat	728	712	708
gap9c4_10x30C.dat	733	723	721
gap9c5_10x30C.dat	733	706	704
gap10c1_10x40C.dat	976	958	955
gap10c2_10x40C.dat	989	963	959
gap10c3_10x40C.dat	977	960	956
gap10c4_10x40C.dat	970	947	942
gap10c5_10x40C.dat	980	947	944
gap11c1_10x50C.dat	1162	1139	1137
gap11c2_10x50C.dat	1194	1178	1176
gap11c3_10x50C.dat	1203	1195	1192
gap11c4_10x50C.dat	1192	1171	1169
gap11c5_10x50C.dat	1195	1171	1168
gap12c1_10x60C.dat	1459	1451	1447
gap12c2_10x60C.dat	1463	1449	1445
gap12c3_10x60C.dat	1450	1433	1428
gap12c4_10x60C.dat	1461	1447	1443
gap12c5_10x60C.dat	1460	1446	1442

Średni błąd dla wyników optymalizacji zamieszczonych w tabeli 1 jest równy 0,28%, co wydaje się wynikiem zupełnie zadawalającym, zwłaszcza że podczas wszystkich optymalizacji czas obliczeń nie przekraczał wartości 0,1 sekundy na PC z procesorem 1 GHz i 1 Gb.

Literatura

[1] Glover F., Taillard E., de Werra D.: *A user's guide to Tabu Search*. Annals of Operations Research, 41:3–28, 1993
[2] Martello S., Toth P.: *An algorithm for the generalized assignment problem*. in: J. P. Brans, ed., Operational Research'81, North-Holland, Amsterdam, 1981
[3] Martello S., Toth P.: *Linear assignment problems*. Annals of Discrete Mathematics, 259–282, 1987
[4] Osman I. H.: *Heuristic for the generalized assignment problem: simulated annealing and tabu search approaches*. OR Spektrum, Vol. 17, 211–225, 1995
[5] OR-library, http://mscmga.ms.ic.ac.uk/