ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2008
Seria: AUTOMATYKA z. 150 Nr kol. 1796

Igor KIERKOSZ, Maciej ŁUCZAK
Politechnika Koszalińska

ALGORYTM PODZIAŁU I OGRANICZEŃ DLA PROBLEMU ROZKROJU NIEGILOTYNOWEGO

Streszczenie. W pracy przedstawiono algorytm optymalizacji rozkroju prostokątnej płyty na szereg prostokątnych elementów przy założeniu cięcia niegilotynowego oraz ograniczeniu na liczbę powtórzeń danego typu elementów w generowanych wzorach rozkroju. W proponowanym algorytmie przeszukiwanie przestrzeni dopuszczalnych rozwiązań odbywa się w oparciu o metodę podziału i ograniczeń. W pracy zamieszczono również wyniki obliczeń dla przykładowych zadań rozkroju dwuwymiarowego.

A BRANCH AND BOUND ALGORITHM FOR NON-GUILLOTINE CUTTING STOCK PROBLEM

Summary. The paper presents an algorithm for two-dimensional non-guillotine cutting stock problem. The problem consists in cutting many rectangular pieces, from a single rectangular sheet in such a way that the amount of trim loss is minimized. Moreover, there is a constraint on the maximum number of each type of piece that is to be produced. The proposed algorithm is based on a branch and bound method. Numerical examples to illustrate the proposed algorithm are solved.

1. Wprowadzenie

Rozważany w pracy problem polega na rozkroju prostokątnej płyty (arkusza) na szereg prostokątnych elementów przy założeniu cięcia niegilotynowego. W rozkroju tego typu nie ma wymogu (w przeciwieństwie do rozkroju gilotynowego), aby kolejne cięcia przebiegały przez całą długość rozcinanego materiału. Celem optymalizacji jest minimalizacja odpadu powstającego w procesie rozkroju lub bardziej ogólnie – maksymalizacja wartości wycinanych elementów.

Problem optymalizacji rozkroju oraz problemy jemu równoważne występują w wielu zastosowaniach praktycznych, na przykład: rozkrój stali, szkła, drewna, papieru; pakowanie palet, pudełek; problem rozmieszczenia ogłoszeń w czasopismach. Oczywiście, w różnych zastosowaniach, w zależności od specyfiki zadania, możemy mieć do czynienia z różnymi ograniczeniami oraz różnymi kryteriami oceny rozwiązań. Omówione tu zagadnienie, w myśl nowej typologii zaproponowanej w [15], można zaliczyć do problemów typu 2SLOPP (*two-dimensional single large*

object placement problems). Dlatego też w dalszej części pracy pojęcie „wycięcie elementu z płyty" będzie utożsamiane z „rozmieszczeniem elementu na płycie".

Zadania optymalizacji rozkroju materiałów oraz wiele innych zadań z zakresu optymalizacji dyskretnej należą do grupy problemów NP-trudnych. W związku z tym w zastosowaniach praktycznych dominują głównie algorytmy przybliżone [1, 7]. Dużą grupę stanowią również metody oparte na metaheurystykach: algorytmach genetycznych i ewolucyjnych [6, 9, 10, 12], algorytmach symulowanego wyżarzania [13]. Niewiele jest natomiast algorytmów dokładnych rozwiązujących to zagadnienie [3, 5]. Jako pierwszy taki algorytm zaproponował w 1985 roku J. E. Beasley [2]. Zastosował on metodę podziału i ograniczeń, w której górne oszacowanie uzyskano dla relaksacji Lagrange'a problemu rozkroju sformułowanego jako zero-jedynkowy problem programowania całkowitoliczbowego.

W niniejszej pracy omówiony zostanie rekurencyjny algorytm generowania dopuszczalnych rozwiązań rozważanego problemu bazujący na opracowanej przez nas metodzie rozmieszczania kolejnych elementów na płycie. Każde rozwiązanie częściowe jest oceniane ze względu na koszt (odpad związany z danym rozwiązaniem) i w zależności od wartości tego kosztu jest rozszerzane o nowe elementy, ewentualnie odrzucane wraz ze wszystkimi możliwymi rozszerzeniami.

2. Opis problemu

Niech dana będzie prostokątna płyta (arkusz) $A=(S,W)$ o szerokości S i wysokości W. Indeksami $i=1,2,\ldots,n$ oznaczmy poszczególne rodzaje części p, które chcemy uzyskać w procesie rozkroju. Zbiór wszystkich typów wycinanych elementów oznaczmy przez P. Niech s_i i w_i będą wymiarami elementu p_i. Każdy sposób rozkroju płyty wyjściowej będziemy nazywać wzorem rozkroju.

Rozcinaną płytę umieśćmy w prostokątnym układzie współrzędnych Oxy, tak aby lewy dolny wierzchołek płyty pokrył się z początkiem układu współrzędnych. Załóżmy ponadto, że:

- mamy do czynienia z rozkrojem niegilotynowym,
- każde cięcie przebiega równolegle do krawędzi arkusza (rozkrój ortogonalny),
- w rozwiązaniu może wystąpić co najwyżej b_i elementów typu p_i (rozkrój z ograniczeniami),
- wycinane elementy mają ustaloną orientację, tzn. nie mogą być obracane o 90°,

Zadanie optymalizacji polegać będzie na maksymalizacji stopnia wykorzystania arkusza wyjściowego (wyrażonego w procentach):

$$\max \rightarrow V=\frac{\sum_{i=1}^{n} a_i s_i w_i}{SW}\cdot 100\% \qquad (1)$$

gdzie: a_i oznacza liczbę elementów p_i występujących w danym wzorze rozkroju, przy czym dla każdego i spełniony musi być warunek: $a_i \le b_i$. Przyjmując takie

kryterium optymalizacji, w łatwy sposób będziemy mogli zdefiniować funkcję kosztu wykorzystywaną w proponowanym algorytmie do oceny bieżących rozwiązań. Koszt danego rozwiązania będzie, mianowicie, równy sumarycznemu polu powierzchni odpadów powstających w procesie rozkroju.

W ogólniejszym przypadku maksymalizacji podlega wartość wycinanych elementów:

$$\max \rightarrow g = \sum_{i=1}^{n} a_i v_i \tag{2}$$

gdzie v_i oznacza wartość elementu p_i.

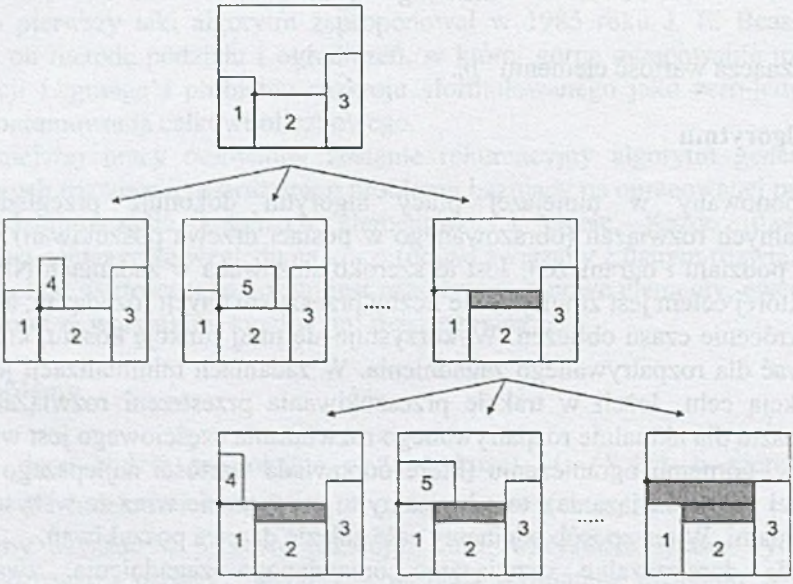
3. Opis algorytmu

Proponowany w niniejszej pracy algorytm dokonuje przeglądu zbioru dopuszczalnych rozwiązań (obrazowanego w postaci drzewa poszukiwań) w oparciu o metodę podziału i ograniczeń. Jest to szeroko stosowana w zadaniach NP-trudnych metoda, której celem jest zmniejszenie liczby przeszukiwanych rozwiązań, a co za tym idzie – skrócenie czasu obliczeń. Wykorzystuje się tutaj funkcję kosztu, którą można zdefiniować dla rozpatrywanego zagadnienia. W zadaniach minimalizacji jest ona na ogół funkcją celu. Jeżeli w trakcie przeszukiwania przestrzeni rozwiązań wartość funkcji kosztu dla aktualnie rozpatrywanego rozwiązania częściowego jest większa lub równa tzw. górnemu ograniczeniu (które odpowiada wartości najlepszego znalezionego do tej pory rozwiązania), to odrzucamy to rozwiązanie wraz ze wszystkimi jego rozszerzeniami. W ten sposób odcinamy całe gałęzie drzewa poszukiwań.

Każde dopuszczalne rozwiązanie omawianego zagadnienia, zwane dalej *rozkrojem,* ma postać listy, składającej się z kolejno rozmieszczanych na płycie elementów oraz tzw. *uzupełnień*, czyli pustych miejsc w rozkroju będących odpadem. Przed umieszczeniem każdego elementu na płycie wyjściowej wyznaczany jest tzw. *punkt zaczepienia,* czyli punkt, w którym znajdzie się dolny lewy wierzchołek rozmieszczanego elementu. Punkt ten jest najbardziej na lewo wysuniętym wśród najniżej położonych punktów obszaru roboczego płyty wyjściowej, tj. obszaru, na którym nie rozmieszczono jeszcze żadnych elementów (czarny punkt na rysunku 1).

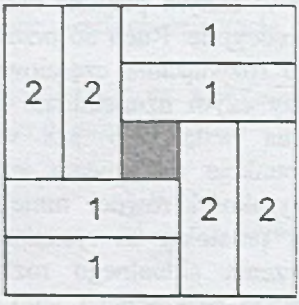
Zbiór rozkrojów można zobrazować w postaci drzewa, w którym korzeniem jest rozkrój pusty, natomiast węzły j-tego poziomu odpowiadają możliwym wyborom elementów listy wstawionych w danym punkcie zaczepienia. Przeszukiwanie tego drzewa dokonywane jest rekurencyjnie. Ruch do przodu (w dół drzewa poszukiwań) jest rozszerzeniem aktualnego rozwiązania częściowego o kolejny rozmieszczany element lub uzupełnienie, przy czym uzupełnienie dodawane jest na końcu, gdy w danym punkcie zaczepienia wstawiono już wszystkie możliwe elementy. Uzupełnienie w aktualnym punkcie zaczepienia jest to prostokąt o największej możliwej szerokości oraz wysokości równej mniejszej z wysokości elementów sąsiadujących z nim (ciemny prostokąt na rysunku 1). Jeżeli natomiast nie ma możliwości dalszego rozszerzenia aktualnego rozkroju, następuje krok w tył w drzewie rekurencji oraz próba rozszerzenia tego rozwiązania o nieanalizowane wcześniej elementy. Krok do tyłu wykonujemy również w przypadku, gdy wartość funkcji kosztu (odpad) dla aktualnego rozkroju jest większa lub równa wartości tej

funkcji dla najlepszego znalezionego do tej pory rozwiązania. W ten sposób w drzewie poszukiwań odcinamy wszystkie gałęzie, które można by uzyskać, rozszerzając aktualne rozwiązanie częściowe. W proponowanym algorytmie koszt danego rozwiązania jest równy sumie powierzchni uzupełnień (odpadów) powstających podczas rozmieszczania elementów na płycie. Przykładowo, kosztem rozwiązania częściowego przedstawionego na rysunku 1 jest suma pól powierzchni ciemnych obszarów.



Rys. 1. Fragment drzewa poszukiwań

Dodawanie uzupełnień do rozkroju na końcu każdego poziomu rekurencji pozwala znaleźć rozkroje, niemożliwe do uzyskania przez niektóre algorytmy znane z literatury [1, 9, 11]. Przykład takiego rozkroju mamy na rysunku 2 (przy ograniczeniach powyżej 4 dla każdego typu elementów). Jednocześnie procedura odcinania powoduje, że dodawanie uzupełnień nie ma znacznego wpływu na zwiększenie liczby przeszukiwanych gałęzi.



Rys. 2. Przykład wzoru rozkroju niemożliwego do uzyskania przez niektóre algorytmy

Pseudokod algorytmu:

Zmienne globalne:
`Elementy` – lista elementów do rozkroju wraz z informacją o ograniczeniach;
`MinOdpad` – minimalny odpad startowy oraz wynikowy;
`MaxRozkrój` – rozkrój wynikowy;

```
01  procedure Algorytm(rozkrój, odpad)
02    punktZaczepienia := aktualny punkt zaczepienia w danym rozkroju;
03    foreach element in Elementy do
04       if element można umieścić w punktZaczepienia then
05          nowyRozkrój := rozkrój plus element;
06          nowyOdpad := odpad;
07          Algorytm(nowyRozkrój, nowyOdpad);
08       endif;
09    uzupełnienie := prostokąt uzupełniający rozkrój w aktualnym punkcie zaczepienia;
10    nowyRozkrój := rozkrój plus uzupełnienie;
11    nowyOdpad := odpad + pole powierzchni prostokąta uzupełnienie;
12    if nowyOdpad >= MinOdpad then return;
13    if nowyRozkrój wypełnia całą płytę then
14       MaxRozkrój := nowyRozkrój;
15       MinOdpad := nowyOdpad;
16       return;
17    endif;
18    Algorytm(nowyRozkrój, nowyOdpad);
19  end.
```

Komentarz do pseudokodu:
(02) Aktualny punkt zaczepienia w danym rozkroju znajdujemy, minimalizując współrzędną y, a następnie współrzędną x powierzchni płyty niezajętej przez rozkrój (czarny punkt na rys. 1).
(04) Element (prostokąt) można umieścić w danym punkcie zaczepienia, jeśli nie ma części wspólnej z elementami aktualnego rozkroju (nie licząc ich brzegów) oraz nie zostało przekroczone ograniczenie liczby elementów danego typu.
(09) Prostokąt uzupełniający rozkrój w aktualnym punkcie zaczepienia jest to prostokąt o największej możliwej szerokości oraz wysokości równej mniejszej z wysokości elementów sąsiadujących z nim (ciemny prostokąt na rysunku 1).
(12) W tym miejscu odcinane są gałęzie reprezentujące rozwiązania, w których na pewno odpad jest nie mniejszy od odpadu najlepszego, dotychczas znalezionego rozwiązania.
(13-17) Zapamiętywane jest najlepsze znalezione rozwiązanie.

Wywołanie procedury:

```
Elementy := aktualna lista elementów do rozkroju;
MinOdpad := pole powierzchni całej płyty do rozkroju;
rozkrójStartowy := pusty rozkrój, bez jakichkolwiek elementów i uzupełnień;
Algorytm(rozkrójStartowy, 0);
```

W wyniku wywołania procedury `Algorytm` w zmiennej `MaxRozkrój` znajduje się wynikowy rozkrój o najmniejszym znalezionym odpadzie, a w zmiennej `MinOdpad` wartość tego odpadu.

4. Eksperymenty obliczeniowe

Powstały algorytm zaimplementowano w środowisku Microsoft Visual Studio (w języku C#) oraz przetestowano dla szeregu zadań testowych zaczerpniętych z literatury oraz z dostępnej w internecie [8] biblioteki zadań testowych. Eksperymenty obliczeniowe przeprowadzono na komputerze z procesorem AMD Athlon 64 3000+. Wyniki obliczeń zamieszczono w tabeli 1. Zadania 1-12 pochodzą z pracy [2], 13-14 z [5], 15 z [14], a 16 z [4]. W stosunku do oryginalnej postaci zadań dokonano modyfikacji kryterium optymalizacji. W związku z przyjętym w algorytmie sposobem określania kosztu rozwiązania założono, mianowicie, że maksymalizacji podlegać będzie stopień wypełnienia (wykorzystania) arkusza obliczany ze wzoru (1), a nie wartość wycinanych elementów (wzór (2)). Dla większości zadań testowych znane są z literatury wartości (ze względu na oba kryteria) rozwiązań optymalnych.

Tabela 1

Wyniki eksperymentów obliczeniowych

Numer zadania	Wymiary arkusza (S, W)	Liczba typów elementów (n)	Uzyskana wartość V	Warunek optymalności	Czas obliczeń w sekundach
1	(10, 10)	5	95	Tak	0
2	(10, 10)	7	97	Tak	0
3	(10, 10)	10	100	Tak	0
4	(15, 10)	5	92	Tak	0
5	(15, 10)	7	93,33	Tak	0
6	(15, 10)	10	100	Tak	0,02
7	(20, 20)	5	43,75	Tak	0,22
8	(20, 20)	7	95	Tak	0,03
9	(20, 20)	10	97,50	Tak	0,42
10	(30, 30)	5	97,67	Tak	0
11	(30, 30)	7	93,56	Tak	0,08
12	(30, 30)	10	99,78	Tak	0,33
13	(30, 30)	7	84,56	Tak	0,02
14	(30, 30)	15	89,67	?	17,97
15	(70, 40)	19	97,36	Tak	0,73
16	(40, 70)	20	97,36	Tak	0,17

Jak widać w tabeli 1, omówiony algorytm dla wszystkich zadań testowych znalazł rozwiązanie optymalne, ewentualnie (zad. 14) rozwiązanie o najwyższej znanej z literatury wartości wypełnienia arkusza. Tylko dla tego zadania czas obliczeń przekroczył 1 sekundę.

Przeprowadzono również eksperymenty obliczeniowe dla zestawu zadań pochodzących z biblioteki [8] (zadania gcut01-12), dla których jednak nie udało nam się w literaturze znaleźć wartości rozwiązań optymalnych ze względu na przyjęte tutaj kryterium. Rozwiązania takie są znane przy założeniu rozkroju gilotynowego ewentualnie rozkroju niegilotynowego z kryterium optymalizacji określonym wzorem (2). Również dla tych zadań proponowany przez nas algorytm w krótkim czasie (od 0 do maksymalnie 5,55 s) generował rozwiązania o wartościach równych lub wyższych od wartości rozwiązań optymalnych podawanych dla rozkroju gilotynowego.

5. Podsumowanie

Zadania optymalizacji rozkroju materiałów, jak wiele innych zadań z zakresu optymalizacji dyskretnej, należą do grupy problemów NP-trudnych. Ze względu na dużą złożoność obliczeniową tego typu problemów oraz zastosowania praktyczne celowe jest opracowywanie oraz rozwijanie efektywnych algorytmów obliczeniowych. W niniejszej pracy omówiliśmy prosty w implementacji algorytm optymalizacji rozkroju niegilotynowego. Co prawda, istnieją wzory rozkroju, których nie można uzyskać, stosując opisaną przez nas metodę rozmieszczania elementów na płycie, jednak we wszystkich zadaniach testowych, dla których znane są rozwiązania optymalne, algorytm je znajdował. Algorytm nie miał również problemu z wyznaczeniem wzorów rozkroju podanych w pracy [6], niemożliwych do uzyskania przez znany z literatury heurystyczny algorytm BL (Bottom Left Algorithm) opisany w [1, 9] oraz algorytm podany w [11].

W dalszych etapach badań planujemy wykorzystać omówiony algorytm jako operator optymalizacji lokalnej w opracowywanym przez nas algorytmie ewolucyjnym, mogącym znaleźć zastosowanie w optymalizacji zadań rozkroju o większym rozmiarze.

BIBLIOGRAFIA

1. Baker B. S., Coffman Jr. E. G., Rivest R. L.: Orthogonal packing in two dimensions. SIAM Journal of Computing, vol. 9, 1980, p. 846-855.
2. Beasley J. E.: An exact two-dimensional non-guillotine cutting tree-search procedure. Operations Research vol. 33, 1985, p. 49-64.
3. Caprara A., Monaci M.: On the two-dimensional knapsack problem. Operations Research Letters. vol. 32, 2004, p. 2-14.
4. Christofides N., Whitlock C.: An algorithm for two-dimensional cutting problems. Operations Research, Vol. 2, 1977, p. 30-44
5. Hadjiconstantantinou E., Christofides N.: An exact algorithm for general, orthogonal, two-dimensional knapsack problems, European Journal of Operational Research, vol. 83, 1995, p. 39-56.
6. Hadjiconstantantinou E., Iori M.: A hybrid genetic algorithm for the two-dimensional single large object placement problem. European Journal of Operational Research, vol. 183, 2007, p. 1150-1166.

7. Hassler R. W., Sweeney P. E.: Cutting stock problems and solution procedures. European Journal of Operational Research, vol 54, 1991, p. 141-150.
8. http://people.brunel.ac.uk/~mastjjb/jeb/info.html
9. Jakobs S.: On genetic algorithms for the packing of polygons. European Journal of Operational Research, vol 88, 1996, p. 165-181.
10. Kierkosz I.: Ewolucyjny algorytm optymalizacji dwuwymiarowego rozkroju niegilotynowego z ograniczeniami. W: Optymalizacja dyskretna. Robotyka i sterowniki programowalne, pod. red. R. Gessinga, T. Szkodnego, WNT, Warszawa 2004.
11. Kierkosz I.: Optymalizacja rozkroju niegilotynowego metodą przeszukiwania z powrotami. Materiały XXV Ogólnopolskiej Konferencji "Polioptymalizacja i komputerowe wspomaganie projektowania", Mielno 2007.
12. Lai K. K., Chan W. M.: An evolutionary algorithm for the rectangular cutting stock problem. International Journal of Industrial Engineering, vol 7, 1997, p. 130-139.
13. Leung T. W., Yung C. H., Troutt M. D.: Applications of genetic search and simulated annealing to the two-dimensional non-guillotine cutting stock problem. Computers & Industrial Engineering, vol. 40, 2001, p. 201-214.
14. Wang P. Y.: Two algorithms for constrained cutting stock problem. Operations Research, vol. 31, 1983, p. 573-586.
15. Wäscher G., Haußner H., Schumann H.: An improved typology of cutting and packing problems, European Journal of Operational Research, vol 183, 2007, p. 1109-1130.

Recenzent: Prof. dr hab.inż. Adam Janiak

Abstract

The paper presents a recursive algorithm for two-dimensional non-guillotine cutting stock problem. The problem consists in cutting many rectangular pieces, from a single rectangular sheet, such that the piece edges are always parallel or orthogonal to the stock rectangle. The objective is to maximize the total area of the pieces cut. We consider the case where there is a maximum number of times that a piece may be used in a cutting pattern.

We proposed a placement procedure and tree search algorithm based on a branch and bound method. Each cutting pattern is represented by vector which informs what types of elements will be considered in a given pattern. At each step, a branch and bound algorithm uses the ranking function to compared the cost (local trim loss) of the current partial solution with total cost of the best solution found so far. If the amount of local trim loss is higher than the best solution cost, the partial solution is abandoned with any possible expansions.

Numerical examples to illustrate the proposed algorithm are solved.