ALGORYTM TABU DLA PROBLEMU PALETYZACJI

Jarosław PEMPERA

Streszczenie: W pracy zaproponowano nowy algorytm oparty na metodzie przeszukiwania z zabronieniami dla problemu paletyzacji. Przedstawiono opis matematyczny problemu oraz zestaw warunków charakteryzujących dopuszczalne rozmieszczenie pudełek na palecie. Przedstawiono wybrane konstrukcyjne algorytmy blokowe oraz główne elementy proponowanego algorytmu tj. definicję zbioru ruchów, wieloetapowy sposób generowania rozwiązań sąsiednich oraz mechanizm zabronień. Przedstawiono wyniki eksperymentu komputerowego mającego na celu ocenę zaproponowanego algorytmu. Testy zostały przeprowadzone na trudnych instancjach literaturowych.

Słowa kluczowe: problem paletyzacji, optymalizacja, algorytm tabu.

1. Wstęp

Współczesne systemy transportowe umożliwiają w krótkim czasie transport na duże odległości (w tym międzykontynentalne) różnego rodzaju towarów. Najczęściej realizowany jest on przez tzw. transport kombinowany obejmujący różne środki transportu takie jak: samochody, pociągi, statki, samoloty. Sprawne przeładowywanie towarów pomiędzy środkami transportu (najczęściej za pośrednictwem centrów logistycznych) wymaga międzynarodowej standaryzacji palet, na których przewożone są dobra. Do rozmiarów obecnie powszechnie stosowanej w Europie europalety przystosowane są rozmiary między innymi ciężarówki.

Koszty transportu towarów zależą od odległości, na którą są przewożone oraz od ich objętości (liczonej ilością palet). Dlatego klientom firm transportowych zależy na upakowaniu jak największej liczby opakowań z transportowanym towarem na palecie. W transporcie towarów na duże odległości dominują przewozy, w których pojedyncze palety wypełnione są identycznymi produktami w identycznych opakowaniach, np. przewozy pomiędzy: producentem i centrum logistycznym, centrami logistycznymi, centrum i dużą jednostką sprzedaży detalicznej.

Problem paletyzacji jest przypadkiem szczególnym dwuwymiarowego problemu pakowania. Badania nad tym problemem rozpoczęły się już w latach 60-tych ubiegłego wieku [1],[2]. Opracowano algorytmy dokładne [3-5], jednakże zastosowanie ich w praktyce ze względu na czas obliczeń ogranicza się do rozwiązywania problemów o niewielkiej liczbie pudełek. W praktyce najczęściej stosuje się heurystyczne algorytmy konstrukcyjne oparte na rozmieszczeniu blokowym [6]. Najefektywniejszy z tego typu algorytm (4–blokowy) pozwala na uzyskanie bardzo dobrych rozwiązań (najczęściej optymalnych) w czasie liczonym w milisekundach na współczesnych komputerach osobistych. W celu uzyskania jeszcze lepszych rozwiązań stosuje się algorytmy oparte na metodach przeglądania przestrzeni rozwiązań takich jak przeszukiwanie z zabronieniami [7] czy przeszukiwanie genetyczne [8].

2. Sformułowanie problemu

W problemie paletyzacji należy rozmieścić zadaną ilość produktów zapakowanych w identyczne prostopadłościenne opakowania (pudełka) na jak najmniejszej liczbie identycznych prostokątnych palet. Produkty w każdym opakowaniu rozmieszczone są w identyczny sposób. Podczas transportu musi być zachowana pionowa orientacja zapakowanych przedmiotów. Z tego względu w pudełku możemy wyróżnić ścianę (powierzchnię), którą będziemy nazywali podstawą.

Z geometrycznego punktu widzenia opakowanie możemy opisać przy pomocy trójki (w,l,h), gdzie w (weight) i l (length) oznaczają szerokość i długość podstawy, natomiast h (high) wysokość. Dalej, niech W oraz L oznaczają odpowiednio szerokość i długość palety, natomiast H maksymalną wysokość sterty paczek. Łatwo można zauważyć, że rozwiązanie problemu paletyzacji sprowadza się do wyznaczenia rozmieszczenia jak największej liczby pudełek na pojedynczej warstwie. Mnożąc tę liczbę przez maksymalną liczbę warstw $\lfloor H/h \rfloor$ (symbol $\lfloor x \rfloor$ oznacza maksymalną liczbę całkowitą mniejszą od x), możemy obliczyć maksymalną liczbę pudełek, które możemy umieścić na palecie oraz liczbę palet potrzebnych do załadowania paczek. W dalszej części pracy, będziemy zajmowali się problemem wyznaczenia rozmieszczenia pudełek na pojedynczej warstwie.

Dla uproszczenia rozważań, bez utraty ogólności, przyjmijmy że $L \geq W$ oraz $l \geq w$, tzn. większy wymiar palety (pudełka) będziemy nazywali długością. Będziemy poszukiwali takiego rozmieszczenia, w którym boki pudełek są równoległe do boków palety. Jest to prostą konsekwencją znanej własności rozwiązania optymalnego rozmieszczenia prostokątów o dowolnych rozmiarach. Położenie pudełka i w warstwie możemy jednoznacznie opisać przy pomocy trójki (x_i,y_i,o_i), gdzie x_i oraz y_i oznaczają współrzędne lewego dolnego rogu pudełka względem lewego dolnego rogu palety, natomiast o_i oznacza orientacje pudełka. Wartość 0, oznacza orientację horyzontalną (dłuższy bok pudełka jest równoległy do dłuższego boku palety), natomiast 1, oznacza orientację wertykalną (dłuższy bok pudełka jest równoległy do krótszego boku palety). Współrzędne pozostałych rogów wynoszą : (x_i+l,y_i), (x_i,y_i+w), (x_i+l,y_i+w) dla orientacji horyzontalnej oraz (x_i+w,y_i), (x_i,y_i+l), (x_i+w,y_i+l) dla orientacji wertykalnej.

Niech $R=\{(x_i,y_i,o_i)\colon i=1,\ldots,b\}$ będzie zbiorem opisującym rozmieszczenie b pudełek. Rozmieszczenie R jest dopuszczalne, jeżeli obrysy pudełek nie wychodzą poza obrys palety oraz nie nachodzą na siebie nawzajem. Formalnie muszą być spełnione następujące warunki:

$$(l \leq L \wedge w \leq W) \vee (w \leq L \wedge l \leq W) \qquad i=1,\ldots,b, \tag{1}$$

spełnienie warunku (1) oznacza, że na palecie możemy umieścić co najmniej jedno pudełko. Niech $x_i' = x_i + l$, $y_i' = y_i + w$ dla $o_i = 0$ oraz $x_i' = x_i + w$, $y_i' = y_i + l$ dla $o_i = 1$, pary (x_i',y_i'), (x_i',y_i), (x_i,y_i') oznaczają współrzędne prawego górnego rogu pudełka i pozostałych rogów.

$$0 \leq x_i \wedge 0 \leq y_i \qquad i=1,\ldots,b, \tag{2}$$

$$x_i' \leq L \wedge y_i' \leq W \qquad i=1,\ldots,b, \tag{3}$$

Warunki (2) i (3) gwarantują umieszczenie pudełek wewnątrz obrysu palety.

$$(x_i \geq x_k') \vee (x_i' \leq x_k) \vee (y_i \geq y_k') \vee (y_i' \leq y_k) \qquad i \neq k,\ i,k=1,\ldots,b, \tag{4}$$

Spełnienie warunku (4) dla każdej pary pudełek gwarantuje drugi z w/w wymogów.

3. Blokowe algorytmy konstrukcyjne

W rozmieszczeniu blokowym pudełek, pudełka zgrupowane są w grupy zwane blokami. Układ pudełek z bloku k tworzy prostokąt składający się z c_k kolumn i r_k wierszy jednakowo zorientowanych i przylegających do siebie pudełek. Oczywiście obrys bloku stanowi prostokąt o długości $c_k l$ oraz o szerokości $r_k w$ w przypadku orientacji horyzontalnej pudełek. Wymiary prostokąta w przypadku orientacji wertykalnej wynoszą odpowiednio $c_k w$ oraz $r_k l$. Rozmieszczenie pudełek w bloku możemy jednoznacznie opisać przy pomocy piątki $B_i=(x_i,y_i,o_i,r_i,c_i)$. Rozmieszczenie blokowe jest dopuszczalne, jeżeli spełnione są warunki analogiczne do warunków (1-4).

Algorytm jedno-blokowy

Na palecie możemy umieścić blok o orientacji horyzontalnej pudełek składający się z maksymalnie $c_h=\lfloor L/l \rfloor$ kolumn oraz $r_h=\lfloor W/w \rfloor$ wierszy. W przypadku orientacji wertykalnej maksymalne liczby kolumn i wierszy wynoszą odpowiednio $c_w=\lfloor L/w \rfloor$ oraz $r_w=\lfloor W/l \rfloor$. Zatem najlepsze rozwiązanie składające się tylko z jednego bloku składa się z $\max(c_h \cdot r_{h,}\, c_w \cdot r_w)$ pudełek.

Algorytm cztero-blokowy

Algorytm cztero-blokowy generuje rozwiązanie składające się z 4 bloków. Każdy z bloków dosunięty jest do innego rogu palety. Zasada działania algorytmu polega na wygenerowaniu wszystkich dopuszczalnych kombinacji liczby wierszy i kolumn dla każdego z bloków i wyboru rozmieszczenia o największej liczbie pudełek. Oczywiście, podczas zmiany liczby wierszy i kolumn dla pewnego bloku, maksymalne rozmiary innych bloków ulegają ograniczeniu. Schemat działania algorytmu cztero−blokowego został przedstawiony na Rysunku 1.

1. $x_1=0,\ y_1=0,\ x_2'=L,\ y_2=0,\ x_3'=L,\ y_3'=W,\ x_4=0,\ y_4'=W,$
2. for $r_1=0,\ldots,\lfloor W/l \rfloor$
3. for $c_1=0,\ldots,\lfloor L/w \rfloor$
4. $r_2=\lfloor (W-y_1')/w \rfloor;$
5. $c_4=\lfloor (L-x_1')/l \rfloor;$
6. for $c_2=\lceil x_1'/w \rceil,\ldots,\lfloor L/l \rfloor$
7. $c_3=\lfloor (L-x_2')/w \rfloor$
8. for $r_3=\lceil (y_2'-y_2)/l \rceil,\ldots,\lfloor W/l \rfloor$
9. $r_4=\lfloor y_3/w \rfloor;$
10. update best solution

Rys. 1. Schemat algorytmu cztero−blokowego

3. Algorytm oparty na metodzie przeszukiwania z zabronieniami TS

Liczne badania eksperymentalne wskazują, że jedną z najbardziej efektywnych metod konstruowana algorytmów dla problemów kombinatorycznych jest metoda przeszukiwania z zabronieniami (tabu). Działanie algorytmów skonstruowanych w oparciu o tę metodę polega na iteracyjnym generowaniu zbioru rozwiązań sąsiednich rozwiązania bieżącego, które w następnej iteracji zastępowane jest najlepszym rozwiązaniem z sąsiedztwa. Rozwiązania sąsiednie generowane są najczęściej przez drobną modyfikację rozwiązania bieżącego.

Elementem charakterystycznym metody jest pewnego rodzaju pamięć odwiedzonych obszarów, która zapobiega powrotowi do rozwiązań wcześniej odwiedzonych. Efektywność algorytmu skonstruowanego w oparciu o tę metodę w znacznej mierze zależy właśnie od tych dwóch elementów.

3.1. Otoczenie rozwiązania blokowego

Rozwiązania sąsiednie rozwiązania blokowego można generować poprzez zmianę położenia bloków, zmianę liczby kolumn i wierszy w blokach, dodanie nowych bloków. Oczywistym jest, że zmiana położenia bloków (bez zmiany ich rozmiarów) nie zmienia liczby pudełek, zatem tego typu modyfikacje możemy odrzucić jako nie dające szansy na wygenerowanie lepszego rozwiązania. Dodawanie nowych bloków możliwe jest tylko wtedy, gdy jest wolne miejsce. Z oczywistych względów pojawia się ono podczas usuwania wierszy i/lub kolumn w blokach (redukcji bloku), jednakże pojawia się również podczas zwiększania liczby wierszy i/lub kolumn (ekspansji bloku) w wyniku usunięcia pudełek kolidujących.

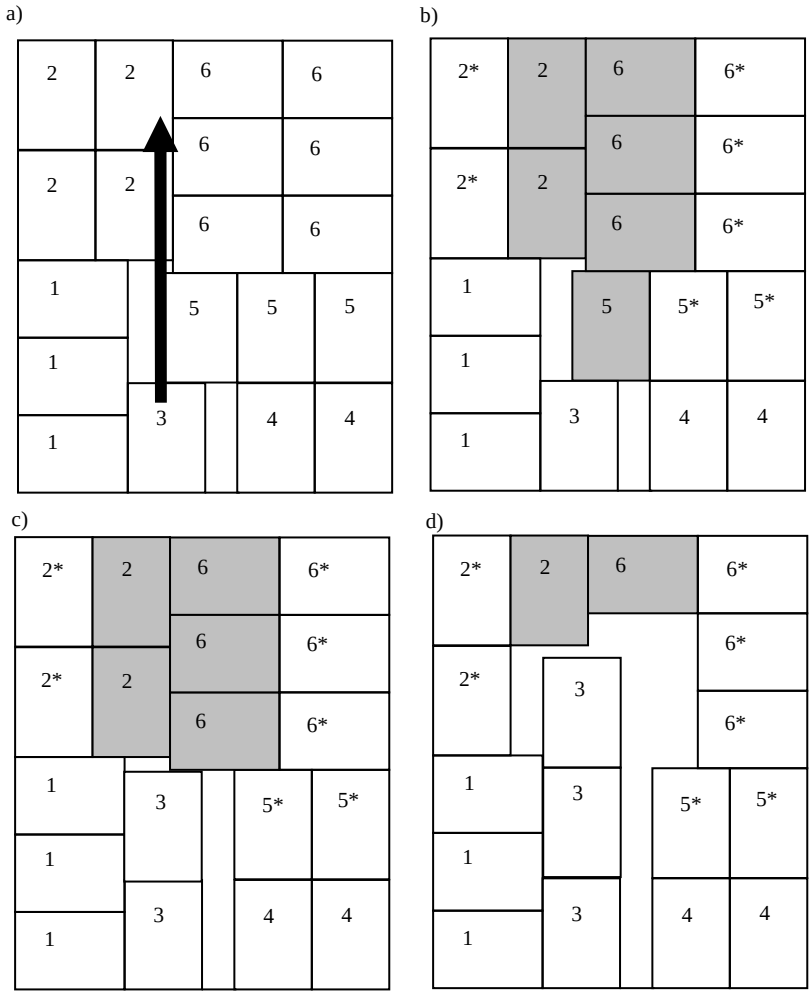
Przed przystąpieniem do precyzyjnego opisania redukcji, ekspansji oraz dodawania bloku zauważmy, że rozważania możemy ograniczyć do modyfikacji tylko w jednym wybranym kierunku. Modyfikacje w pozostałych kierunkach możemy wykonać w ten sam sposób obracając wcześniej paletę z rozmieszczonymi pudełkami o odpowiednią wielokrotność kąta prostego. Dla ustalenia uwagi, modyfikacji ulegać będzie liczba wierszy bez zmiany położenia lewego dolnego rogu pudełka (patrz ekspansja bloku 3 na Rysunku 2).

Oba typy ruchów możemy opisać przy pomocy pary (i,x), gdzie i oznacza numer bloku, natomiast x nową liczbę wierszy w bloku. Oczywiście, jeżeli $x<r_i$ wówczas nastąpi redukcja bloku, natomiast dla $x>r_i$ ekspansja bloku. Wartość x ograniczona jest z dołu wartością zero, natomiast z góry wartością $\lfloor (W-y_i)/w \rfloor$ lub $\lfloor (W-y_i)/l \rfloor$ w zależności od orientacji pudełek w bloku. Realizacja redukcji bloku jest trywialna. W dalszej części sekcji skupimy się na omówieniu ekspansji i wypełniania pustych przestrzeni (dodawania nowych bloków).

3.2. Ekspansja bloku

Realizacja ekspansji bloku pomimo prostoty opisu jest czynnością trudną w implementacji, ponieważ wymaga wyznaczenia wszystkich pudełek kolidujących i usunięcia ich. Co więcej najczęściej usunięcie pudełek wymaga rozbicia bloku oraz powtórnego utworzenia jednego lub więcej bloków. Wykonanie wszystkich czynności dla każdego możliwego rozszerzenia wymaga dużo czasu. Czas wyznaczania wszystkich rozszerzeń można znacznie skrócić wykonując wstępnie identyfikację bloków

kolidujących, podział każdego bloku na trzy bloki: (*i*) składający się z kolumn kolidujących z rozszerzanym blokiem, (*ii*) kolumn nie kolidujących znajdujących się z lewej strony kolumn kolidujących, (*iii*) kolumn nie kolidujących znajdujących się z prawej strony. Po wykonaniu tego typu czynności wstępnych, wygenerowania rozwiązania będącego rozszerzeniem bloku wymaga jedynie usunięcia odpowiedniej liczby wierszy (dolnych) jedynie w blokach kolidujących. Kolejne etapy wyznaczenia rozszerzeń bloku 3 przedstawiono na Rysunku 2. Kierunek ekspansji bloku pokazano na Rysunku 2a, podział bloków kolidujących (2, 5 i 6) na Rysunku 2b (gwiazdką zaznaczono fragmenty bloków nie kolidujących z rozszerzanym), natomiast na Rysunkach 2c i 2d rozwiązania wygenerowane przez rozszerzenie bloku odpowiednio o jeden i dwa wiersze.

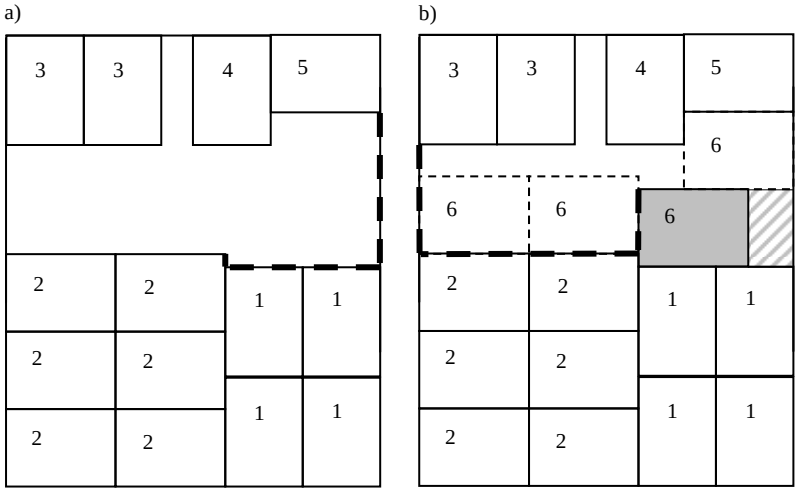


Rys. 2. Ekspansja bloku 3

3.3. Wypełnienie pustych przestrzeni

Zarówno redukcja jak i ekspansja bloków w naturalny sposób powodują powstawanie niewykorzystanych powierzchni palety. W przypadku redukcji, wypełnienie wolnych przestrzeni palety pudełkami o identycznej orientacji jak w redukowanym bloku w oczywisty sposób prowadzi do odtworzenia usuniętych pudełek. W przypadku ekspansji bloków wypełnienie wolnych przestrzeni pudełkami dla każdej orientacji prowadzi do wygenerowania innych rozwiązań.

Algorytm najlepszego wypełnienia (best fit [9]) jest jednym z najlepszych algorytmów konstrukcyjnych do rozwiązywania problemów pakowania 2D. Jednym z najistotniejszych elementów tej metody jest pojęcie tzw. dziury. Jest to wolna przestrzeń na palecie położona najbliżej jej dolnej części. Przestrzeń ta ograniczona jest z obu boków, bokami pudełek i/lub palety, w podobny sposób ograniczona jest od dołu. Dziura jest użyteczna, jeżeli można w niej umieścić pudełko. W dziurze umieszczane jest pudełko, które najbardziej ją wypełnia tj. o największej długości i/lub szerokości nie większej od szerokości dziury. W rozważanym przypadku wszystkie pudełka mają jednakowe rozmiary, zatem realizacja wyboru sprowadza się do wyboru orientacji pudełka. W przypadku, gdy dziura nie jest użyteczna wydzielany jest prostokątny obszar stracony, którego wysokość równa jest wysokości mniejszego z boków. Procedura wyznaczania dziur i ich wypełniania kontynuowana jest do momentu oznaczenia wszystkich obszarów niewykorzystanych. Na Rysunku 3a oraz 3b zilustrowano opisaną metodę. Na Rysunku 3a pokazano pierwotną wolną przestrzeń z zaznaczonym konturem pierwszej dziury (grubą kreskowaną linią), natomiast na Rysunku 3b przedstawiono blok wypełniający dziurę (szary), kontur dziury w następnej iteracji, obszar stracony (zakreskowany) oraz obrysy pudełek wstawianych w kolejnych iteracjach. Wszystkie wstawiane pudełka zostały oznaczone numerem 6, w każdym z przypadków najlepszą orientacją (i/lub jedyną dopuszczalną) pudełek okazała się orientacja horyzontalna.



Rys. 3. Wypełnienie wolnej powierzchni

3.4. Dosunięcie bloków do najbliższych rogów palety

Znaczne rozproszenie niewykorzystanych obszarów palety jest niekorzystne z punktu

widzenia szans na wygenerowanie nowych lepszych rozwiązań. Wynika to z cech przedstawionych ruchów, które modyfikują rozmieszczenie pudełek tylko w niewielkim otoczeniu modyfikowanego bloku. Z tego punktu widzenia najbardziej korzystnym rozmieszczeniem jest rozmieszczenie, w którym niewykorzystane obszary palety znajdują się jak najbliżej siebie, najlepiej w pobliżu środka palety.

Rozmieszczenie mające tego typu korzystne cechy można uzyskać przesuwając bloki do najbliższych rogów palety. W tym celu należy wyznaczyć odległość euklidesową każdego rogu każdego bloku względem każdego rogu palety, wybrać blok o najmniejszej odległości i dosunąć do najbliższego w tym sensie rogu. Dosunięcie bloku można zrealizować wzorowanym na usprawnionym algorytmie dół-lewo (ang. improved bottom left algorithm [10]). W algorytmie tym pudełko (tutaj blok) przesuwane jest maksymalne w dół, następnie przesuwane jest w lewo na taką odległość, która umożliwia przesunięcie w dół. Czynności te powtarzane są w takiej kolejności dopóki następuje przesunięcie pudełka chociażby o jedną jednostkę. Oczywiście przesunięcie pudełka nie może generować rozwiązania niedopuszczalnego tj. sprawdzane jest czy nowe położenie bloku nie koliduje z rozmieszczeniem innych bloków. Przesunięcie bloków w innych kierunkach realizowane jest w podobny sposób.

3.5. Precyzyjny opis zbioru ruchów i sąsiedztwa

Zbiór ruchów $V(B)$ dla rozwiązania blokowego $B=(B_1,...,B_b)$ możemy zdefiniować w następujący sposób:

$$V(B)=\{(i,r,k) : i=1,\ldots,b,\ r=1,\ldots,4,\ k=0,\ldots,g_{ir}\}, \tag{4}$$

gdzie i jest numerem bloku, r kierunkiem ekspansji lub redukcji bloku (1–w górę, 2–w dół, 3–w lewo, 4–w prawo), natomiast g_{ir} maksymalną liczbą wierszy (kolumn) z jakich może się składać blok i po rozszerzeniu w kierunku r.

Wykonanie ruchu $v=(i,r,k)$ powoduje odpowiednią modyfikację bloku i oraz dodatkowo: (*i*) usunięcie kolidujących pudełek, (*ii*) wypełnienie pustych przestrzeni palety, (*iii*) przesunięcie bloków w kierunku najbliższych rogów.

3.6. Lista zabronień

Lista zabronień jest pewnego rodzaju pamięcią przeglądniętych rozwiązań. Głównym celem mechanizmu zabronień bazującego na tej liście jest zabezpieczenie przed generowaniem rozwiązań wcześniej wygenerowanych i w konsekwencji cyklicznego powtarzania trajektorii poszukiwań. W praktyce, ze względu na dużą ilość pamięci niezbędną do zapamiętania kompletnych rozwiązań, na liście zapamiętywane są tylko pewne informacje pochodzące od tych rozwiązań i/lub ruchów je generujących, które będziemy nazywali atrybutami. Lista ta ma ograniczoną długość L wyznaczaną w sposób eksperymentalny.

Na potrzeby realizacji mechanizmu zabronień, każdemu blokowi przyporządkowywany jest na etapie powstawania unikalny identyfikator. Niech $v=(i,r,k)$ będzie ruchem generującym nowe rozwiązanie bazowe modyfikującym blok i o identyfikatorze id. Na liście zabronień zapamiętywana jest para (id,r). Ruch $w=(j,s,k)$ modyfikujący blok o identyfikatorze id zabroniony, jeżeli na liście zabronień pamiętana jest para (s,id). Mechanizm ten zabrania wykonywania jakichkolwiek modyfikacji w wybranym kierunku bloków wcześniej modyfikowanych w danym kierunku. Dla przykładu, po ekspansji pewnego bloku zabronione jest wykonanie redukcji, w konsekwencji odtworzenie liczby

wierszy, przez najbliższe L iteracji. Oczywiście nie wyklucza to redukcji bloku będącego konsekwencją ekspansji innych bloków.

4. Eksperyment komputerowy

Celem badań eksperymentalnych była ocena efektywności algorytmu TS. Eksperyment komputerowy został przeprowadzony, podobnie jak w pracy [11], na 12 instancjach danych testowych. Cztero-blokowy algorytm konstrukcyjny oraz algorytm TS zaimplementowany w języku C++ środowisku Visual Studio 2005. Badania testowe przeprowadzono na komputerze przenośnym HP Mobile Workstation z procesorem Core2 Duo 2,6 GHz. Algorytm TS uruchomiono na 1000 iteracji z długością listy równą 23.

Dla każdej instancji wyznaczono liczbę pudełek otrzymaną algorytmem cztero-blokowym b_{4b} oraz najlepsze rozwiązanie b_{TS} przeglądane podczas przeszukiwań algorytmem TS, ponadto zapamiętano czas t_{TS} jaki upłynął od momentu rozpoczęcia przeszukiwania do momentu znalezienia tego rozwiązania oraz czas t_{1000} wykonania 1000 iteracji. Rezultaty eksperymentu komputerowego zostały przedstawione w Tabeli 1. W dodatkowych dwóch kolumnach przedstawiono wartości optymalne b_{opt} (maksymalne liczby pudełek) zaczerpnięte z pracy [11] oraz współczynnik fact = b_{TS} / t_{1000}. Współczynnik ten ocenia zależność czasu obliczeń od liczby pudełek.

Tab. 1. Rezultaty eksperymentu komputerowego

lp	W	l	W	L	b_{opt}	b_{4b}	b_{TS}	t_{TS}	t_{1000}	*fact*
1	3	5	16	22	23	22	22		11	2,1
2	5	12	44	57	41	40	41	0,1	12	3,4
3	3	4	22	23	42	41	41		11	3,8
4	4	9	39	42	45	43	44	0,98	18	2,5
5	4	9	33	52	47	44	46	0,01	18	2,6
6	5	11	41	64	47	44	47	0,01	18	2,6
7	3	7	25	40	47	45	46	0,13	19	2,5
8	5	12	52	56	48	46	46		12	4
9	3	7	26	43	52	52	52		19	2,7
10	5	17	71	109	90	90	90		45	2
11	6	7	47	87	97	94	95	0,14	53	1,8
12	8	9	85	127	149	145	149	0,37	84	1,8

5. Wnioski

Z danych zawartych w Tabeli 1 wynika, że algorytm TS dla 7–miu instancji wygenerował rozwiązania lepsze od rozwiązań wygenerowanych algorytmem cztero–blokowym oraz w przypadku 5–ciu rozwiązania optymalne. W pozostałych przypadkach rozwiązania wygenerowane przez algorytm TS różnią się nie więcej niż dwoma pudełkami od optymalnych. Różnice pomiędzy rozwiązaniami cztero–blokowymi oraz optymalnymi również nie są duże, w dwóch przypadkach algorytm ten wygenerował rozwiązania optymalne. Oba fakty świadczą o bardzo dużej efektywności tego algorytmu.

Czas wykonania 1000 iteracji algorytmu TS nie przekraczał 90 sekund w każdym przypadku. Współczynnik *fact* maleje wraz ze wzrostem liczby pudełek, które można rozmieścić na palecie. Świadczy to o tym, że dla dużych instancji przetwarzaniu podlegają

bloki składające się z wielu pudełek i w konsekwencji istotnie skracany jest czas wykonania jednej iteracji algorytmu.

6. Podsumowanie

W pracy przedstawiono opis matematyczny problemu paletyzacji, zaproponowano algorytm oparty na metodzie przeszukiwania z zabronieniami. Opisano zbiór ruchów zdefiniowany dla rozwiązań blokowych, sposób konstruowania rozwiązań sąsiednich oraz pewne wskazówki pozwalające na przyspieszenie procesu przeszukiwań. Biorąc pod uwagę fakt, że najlepsze rozwiązanie wygenerowane zostało przez algorytm TS w czasie nie przekraczającym jednej sekundy algorytm ten może być z powodzeniem stosowany w systemach informatycznych i\lub automatyki wspomagających proces paletyzacji wyrobów.

Literatura

1. Gilmore P.C., Gomory R.E.: A Linear Programming Approach to the Cutting Stock Problem, Operations Research, 9, 1961, 849–859.
2. Gilmore P.C., Gomory R.E.: Multistage cutting problems In two Or mode dimensions, Operations Research 12, 1965, 94–119.
3. Cani P.De: Packing problems in theory and practice, Ph.D.Thesis Department of Engineering Production, University of Birmingham, 1979.
4. Bhattacharya R., Roy R., Bhattacharya S.: An exact depth-first algorithm for the pallet loading problem, European Journal of Operational Research, 110, 1998, 610–625.
5. Gustavo H., Robert F.: Solving the pallet loading problem, European Journal of Operational Research, 184, 429–440, 2008.
6. Steudel H.J.: Generating pallet loading patterns: A special case of the two-dimensional cutting stock problem, Management Science 25, 1979, 997–1004.
7. Dowsland K.A.: Simple tabu thresholding and the pallet loading problem.In: Osman, I.H. and Kelly, J.P. (Eds). Metaheuristics: theory and applications, Kluwer Academic Publishers, 1996, 379–405.
8. Herbert A., Dowsland W.: A family of genetic algorithms for the pallet loading problem, Annals of Operations Research 63,1996, 415–436.
9. Whitwell G.: Novel Heuristic and Metaheuristic Approaches to Cutting and Packing**,** Thesis Submitted to the University of Nottingham, 2004, 68–78.
10. Liu D., Teng H.: An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles, European Journal of Operational Research, 112, 1999, 413–420.
11. Morabito R., Morales S.: A simple and effective recursive procedure for the manufacturer's pallet loading problem. Journal of the Operational Research Society 49, 1998, 819–828.

Dr inż. Jarosław PEMPERA
Instytut Automatyki, Informatyki i Robotyki
Politechnika Wrocławska
50-372 Wrocław, ul. Wybrzeże Wyspiańskiego 27
tel./fax.: (71) 320-28-34
e-mail: jaroslaw.pempera@pwr.wroc.pl