The local search algorithm in polynomial neighborhoods for the linear packing problem

A.R. Usmanova
Department of Computer Science and Robotics
Ufa State Aviation Technical University
Ufa, Russia
e-mail: kfmn2004@mail.ru

A.P. Zemlyanov
Department of Computer Science and Robotics
State Aviation Technical University
Ufa, Russia
e-mail: zemlianovap@gmail.com

Annotation[1]

The Bin Packing Problem can be found widely in different branches of industry and technique. The conception of solutions' neighborhoods and the ways of its construction are considered. The undetermined algorithm of local search in the proposed neighborhoods is offered. The computing experiment performed on the difficult benchmark problems taken from the OR-library is proved the effectivity of the proposed way.

1. Introduction

Let us describe the Bin Packing Problem (BPP): the set L={w_1,w_2,…,w_n} of nonnegative weights of items and positive number C – the bin capacity. Without loss of generality let the item weights and bin volume will be integer numbers. It is necessary to find such partition L into the minimal number of disjoint subsets, that the sum of weights in each subset does not exceed the bin capacity C. This is NP-hard problem [1]. If we shall interpret the set L as sizes or length of some items, not as the weights and C as the length of the big object – stock, which will be cut onto smaller items, then we can discuss the linear cutting stock problem. The formal mathematical definitions of these problems, obviously, are identical and using the terms «cutting» or «packing» is the questions of private preferences. In this paper mostly the terminology of bin packing problem will be used.

Let us formulate one of mathematical definitions of BPP [2]. We are given the set L={w_1,w_2,…,w_n} of item weights and n bins of capacity C. Let us assign each item to one and only one bin, i.e. pack item to bin, so that the total weight packed in any bin does not exceed the capacity, and the number of used bins will be minimal. We suppose that the bin containing at least one item is used, otherwise bin is not used. Now let us introduce two binary variables. Let $y_i \in \{0,1\}, i \in N = \{1,2,\ldots,n\}$

where $y_j = \begin{cases} 1, \text{if j - th bin is used} \\ 0, \text{otherwise} \end{cases}$

and let $x_{ij} \in \{0,1\}$ $\text{i, j} \in \text{N}$,

where $x_{ij} = \begin{cases} 1, \text{if i - th item is packed into bin j;} \\ 0, \text{otherwise} \end{cases}$

Then we can formulate BPP as Integer Linear Program (ILP):

Minimize $z = \sum_{j=1}^{n} y_j$

with respect to $\sum_{i=1}^{n} y_j x_{ij} w_i \le C,\ j \in N$

and $\sum_{i=1}^{n} x_{ij} = 1,\ j \in N$.

There are many algorithms for solving this problem both exact and approximate have been proposed at the present. Since NP-hardness of BPP the most interest are the effective approximate algorithms, so as the simplest heuristics and metaheuristics with using genetic and evolutionary algorithms, taboo search method, simulating annealing algorithms and others. One of the simplest heuristics is First Fir method and its online variant for the sorted list of item's weights – First Fit Decreasing. FF algorithm each item pack into the first bin with the fit remaindered capacity. In the case if there are no fit bins then item is packed into a new bin. The algorithm FFD at first orders items in nonincreasing order. If the binary tree is used for storing the data describing bins during the algorithm execution and quick sorting is applied then first fit algorithms requires O(nlogn) time.

In this paper the local search algorithm applied to the initial solutions obtained by the greedy stochastic heuristics that are non-deterministic versions of FFD algorithm and described at [3,4] is considered.

Let the solution of some optimization problem can be defined by vector **x**, the set of all feasible vectors satisfied

Proceedings of the 18th international workshop on computer science and information technologies 18thCSIT'2016, Czech Republic, Prague, Kunovice, 2016

to the problem constraints denote as X. Let it is necessary to maximize some function $f(x)$ on the set of vectors X, i.e. we want to find such vector*, that $f(x^*) \geq f(x)$ for any $x \in X$.
Let applying some operator *op* to the current solution ***x*** lead to the new feasible solution ***y***, denote this fact as *op(**x**)=**y***. Usually applying an operator to the solution generates not one but some set of solutions that we shall name the *neighbourhoods of solutions* and denote as *N(**x**)*, so we have *op(**x**)=**y**, **y**∈N(**x**)*. The simplest algorithm can be described as follows:
Algorithm L

1. To obtain the initial solution $\boldsymbol{x^0}$.
2. To set the current solution $\boldsymbol{x}=\boldsymbol{x^0}$
3. To construct the neighbourhood of solutions *N(**x**)*
4. To search such solution ***y'***∈*N(**x**)* that $f(y') \geq f(y)$, ***y***∈*N(**x**)*
5. To set the current solution ***x***= ***y'***
6. If the halt conditions are satisfied then to terminate the algorithm with result x , otherwise go to step 3.

In this article two ways of neighborhoods constructing are proposed. The first one uses the removing one item from one bin to another. The second is considered the items swapping for different bins. Also there is three ways of such swapping are considered.

The algorithm used for initial solution and just the local search algorithm described below are non-deterministic, depending on some parameter. This allows to reduce the solution browsing in the neighborhood and therefore to increase the performance of the algorithm.

In this paper the numerical experiment devoted to investigation of the algorithm with different parameters and to comparing it with its deterministic version.

2. The simple greedy heuristics

Algorithm RPEP − Random Permutation with Equal Probabilities

This algorithm is the simplest and the most used when it is necessary to obtain different initial solutions for one problem. The set of all permutations of the items with cardinality n! are considered. The element from this set randomly chosen and packed by FF order accordingly the chosen order of items.

The computational complexity of random permutation modeling is $O(n \log n)$, because for modeling of each of vector component *log n* operations are required for searching and removing items from the set N. It does not exceed the computational complexity of FF algorithm, so the complexity of RPEP algorithm coincides with FF and equals to $O(n \log n)$.

Algorithm RPP − Random Permutation with Parameter

As in the previous case algorithm generates some random sequence of item indices and then applies to it FF algorithm. But unlike the first algorithm all possible permutations are not equiprobable. At first items are sorted in decreasing order. Then, during the packing process, considering the next item, algorithm compares the given parameter *p* with the value of the pseudo-random variable uniformly distributed on [0,1]. The item will be packed just in the case if parameter is greater of this random variable. The process continues until all items are packed. When we are considering items, the random events, which are that the choose of the next item, are equiprobable and independent, because that the single scanning of item list is the scheme of Bernoulli trials. Therefore, the single scanning of *n* items gives us the mathematical expectation of packed items of equal to *np*. So to pack all *n* items we will need an average of n*/np=1/p* steps. In that way the computational complexity can be estimated as: it required *1/p* cycles, each of that performs *n* operations of scanning items and *log n* operations for each item when searching for bin. Because the parameter *p* does not depend of the dimension of the problem, the computational complexity coincides with the complexity of FFD and equals to $O(n \log n)$.

Algorithm RBP − Random Bin with Parameter

This algorithm does not modify the original order of the items (sorted by non-increasing weights), but the strategy of choosing the bin for each item becomes non-deterministic. The simplest realizations of such a strategy consists in choosing one of the feasible bins for a given item, with the events consisting in the choice of each bin, are equiprobable and form the collectively exhaustive events. More interesting is the algorithm in which the probabilities of bin choosing are not equiprobable and depend on some parameter. When we execute the assigning of the current item to the bin, we scan the bins with suitable residual capacity in descending order. The probability of that item will be packed to each bin is the parameter of algorithm - $b \in (0,1]$. Under this strategy, the choice of bin for packing is not a certain event, especially if the parameter is close to zero, and a list of feasible bins has a small length. If no bin is chosen, there are several possibilities: to pack the item to any of feasible bins, for example, to the first or to the last; put it in a new bin or leave the item until unpacked and repeat the process until all items will not be assigned to the bins. The advantage of the first strategy is that in this case it is easy to estimate the computational complexity of the algorithm and to predict its behavior for small values of *b*. In the second case, when values of b close to zero, it can result in very bad solutions, where each bin is assigned to a very small number of items and the value of the loss function will be much worse than optimal. It is proposed to give preference to the third strategy, which allows to obtain solutions with a good enough value of the loss function.

The disadvantage is that the computational time of the algorithm will greater than using the first or second strategy. When *b=1* the algorithm is reduced to a deterministic FFD, if after packing of each item bins will be sorted in order of descending its residual capacity. The computational complexity of the described above algorithm is $O(nm) = O(n^2)$.

3. The Local Search Algorithm

The considered above simple heuristics, which has such advantages as simplicity of implementation and small computation time, also possess disadvantages. The most significant of which is the inability to improve the obtained solution. Therefore such algorithms should be combined with other methods if they are applied for the solving optimization problems. In particular, they should used with the various iterative algorithms that successively improve the solution. Here the local search algorithm is proposed, which at each step considers a set of neighboring solutions, the so called neighborhood of the current solution. The solution delivering the maximum of the criterion function is chosen from neighborhood as the next solution. The process continues until there are solutions belonged to the neighborhood better than the current in relation to the criterion function.

The polynomial neighborhoods

To construct a neighborhood of solutions the current solution is exposed some operators The first such operator can be described by a triple *op1(j_1,j_2,i_1)*, indicating that the item i_1 is removed from the bin j_1 to the bin j_2. The set of feasible solutions the can be obtained by using this operator we denote as N_1. In average the number of elements in neighborhood N_1 will be about of $m \cdot m \cdot n/m = m \cdot n$, where m is number of used bins, and *n* is the number of items. The advantage of applying this neighborhood is that it may contain a solution that improves the loss function, that is, reduces the number of bins used. But the disadvantage is that this neighborhood may be empty, when any movement of one item results in an unfeasible solution. Therefore, searching the next solution only at that neighborhood is not always possible, and it is necessary to consider one more set of neighboring solutions.

For this purpose let introduce the second operator *op2(i_1,i_2,j_1,j_2)*, the application of which to the current solution means a swapping of the i_1–th item from j_1–th bin with i_2–th item from j_2–th bin. The second neighborhood N_2 is the set of feasible solutions obtained by the application of op2 to the current solution. In average this neighborhood can contain about $m \cdot m \cdot n/m \cdot n/m = n^2$ elements. Although this neighborhood does not provide solutions with the better values of the lost function than the current solution, but the choosing of solutions from it allows to turn around in the case when the set N1 is empty. In the early stages of the algorithm development the next variants for swapping items are considered:

- a random item from a bin to another random item from the second bin (with equiprobability for the number of items in each bin)
- item with maximal weight from bin to another item with maximal weight from another bin
- item with minimal weight from bin to another item with minimal weight from another bin
- item with maximal weight from bin to another item with minimal weight from another bin

The computing experiment has shown that the best results are obtained when the first or the second way are used. Further, it was decided to apply just the first way to construct a neighborhood.

We consider only feasible solutions in neighborhoods N_1 and N_2, so for each solution the restriction

$$\sum_{i=1}^{n} y_j x_{ij} w_i \le C,\ j \in N$$

holds.

For the time reduction of exhaustive search in neighborhoods at first the search through the neighborhood N_1 is performed, and only if N_1 is empty or if N_1 does not contain solutions better than the current one, the search continues through the neighborhood N_2. So the common neighborhood of solutions can be described as

$$N^* = \begin{cases} N_1, if\ N_1 \ne \varnothing\ and\ \exists y : f(y) > 0 \\ N_2,\ otherwise \end{cases} ,$$

where f is the criterion function.

It is very important for this algorithm to look for a good criterion function because the loss function values coincide for the most of neighboring solutions. We propose as the criterion function $\overline{f} = \Delta_2 - \Delta_1$, where Δ_1 – difference of weights i_1-th and i_2-th bins at the current solution, and Δ_2 – at the next solution, obtained by removing item from bin i_1 into bin i_2. Speaking in more detail these differences are evaluated as

$$\Delta_1 = \left| \sum_{i=1}^{n} x_{ij_1} w_i - \sum_{i=1}^{n} x_{ij_2} w_i \right| \text{ for } N_1 \text{ or } N_2 \text{ and}$$

$$\Delta_2 = \left| \sum_{i=1}^{n} x_{ij_1} w_i - w_{i_1} - \left(\sum_{i=1}^{n} x_{ij_2} w_i + w_{i_1} \right) \right| =$$

$$= \left| \sum_{i=1}^{n} x_{ij_1} w_i - \sum_{i=1}^{n} x_{ij_2} w_i - 2w_{i_1} \right|$$

if we consider the neighborhood N_1, and

$$\Delta_2 = \left| \sum_{i=1}^{n} x_{ij_1} w_i - w_{i_1} + w_{i_2} - (\sum_{i=1}^{n} x_{ij_2} w_i + w_{i_1} - w_{i_2}) \right| =$$
$$= \left| \sum_{i=1}^{n} x_{ij_1} w_i - \sum_{i=1}^{n} x_{ij_2} w_i - 2(w_{i_1} + w_{i_2}) \right|$$

for the neighborhood N_2.

Then at each iteration the next solution $y*$ is such as $f(y*) \geq f(y), \forall y \in N^*$. The advantage of this criterion function is that its computation does not depend on the dimension of the problem. We can say that thanks to this function, the algorithm tends to "load" bins with great weight and "unload" a bin with less weight. This makes it possible to remove all the items from a previously used bin and to improve the value of the loss function.

The halt condition for the proposed local search algorithm is either reaching of lower bound $N_0 = \frac{\sum_{i=1}^{n} w_i}{C}$ (obviously the optimal solution) or there are no in the neighborhood N* the solutions with a positive value of the criterion function. The second case means we have obtained the locally optimal solution. Another halt conditions as the restrictions of performance time or the iterations number were also considered and finally the iterations number restriction was included.

After the obtaining the locally optimal solution we can generate a new initial solution by the algorithms RPEP/RPP/RBP and repeat the local search. Let the number of generations of initial solutions is *l*, then an algorithm allows to obtain *l* locally optimal packings (unless the global optimum is not reached). It is obviously that increasing the parameter *l* increases the probability of obtaining the global optimum. Let us describe the scheme of the algorithm with parameter l.

Algorithm LS

1. To generate an initial solution by RPEP/RPP/RBP algorithm.
2. To search $y*$ such as $f(y*) \geq f(y), \forall y \in N^*$
3. If the loss function *z(y*)=N₀* then to terminate algorithm
4. If $f(y^*) \leq 0$ then step 5, otherwise to set $x = y^*$ and step 2
5. To repeat steps 1-4 *l* times ant to terminate with result *x*

The local search algorithm can also be made probabilistic. For this purpose it is possible to consider not all solutions in the neighborhood, but just some random sampling of solutions from it. Let's add to the algorithm a new parameter *lp* that is the probability with which every solution in N* will be included to this sample. The value of this parameter should be defined experimentally. The introduction of non-determinism in the local search algorithm has a dual purpose: reducing the computing time and obtaining the better (with respect to the loss function) solutions than the solutions provided by the deterministic algorithm. The first purpose is fairly obvious – the number of scanning neighboring solutions at each iteration will decrease by *1/lp* times and, accordingly, the time of solving the problem should decrease. The second purpose reaching can be explained by such arguments: probabilistic search has a less possibilities to be stuck in locally optimal points. So obtaining more perspective solutions with respect to the loss function is provided. It should be noted that the local search algorithm was later used as a part of more complex metaheuristic - tabu search method [6,7]. And there the changing of parameter *p* plays the role of the intensification procedure – p increases, when search performs with improvement of the criterion function, and decreases otherwise.

4. Computing experiment

Benchmarks

We have used 8 benchmark sets from the OR-library proposed by Falkenauer[5], each set has 20 randomly generated problems united by a common characteristic of the source data. The Table 1 gives these characteristics for each set.

Table 1. Characteristics of benchmarks

№ set	n – number of items	C –bin capacity	Item weights
1	120	150	[20,100]
2	250	150	[20,100]
3	500	150	[20,100]
4	1000	150	[20,100]
5	60	100.0	[20.0,50.0]
6	120	100.0	[20.0,50.0]
7	249	100.0	[20.0,50.0]
8	501	100.0	[20.0,50.0]

These benchmarks are quite difficult and the solutions obtained by the simple heuristic algorithms in most cases far from optimal. The best known solutions are given for all sets and were obtained by Falkenauer with a genetic algorithm. In almost all cases, these solutions are obviously optimal, because they coincide with a lower bound. Table 2 gives an average loss function values (for 20 problems) – number of used bins. The benchmarks of sets 5-8 are so-called triplets, an optimal solutions usually there is exactly three items packed into each bin. The triplets of the same set always have the same number of used bin in the obvious optimal solutions.

Table 2. An average values of the best known number of used bins.

№ set	An average value of bin
1	49.15

2	101.70
3	201.20
4	400.55
5	20.00
6	40.00
7	83.00
8	167.00

Computing experiment with determined and probabilistic local search algorithms

This computing experiment was hold with purpose of comparing the determined and probabilistic local search algorithms described above.

The benchmark problems of set 1-8 were solved by the local search algorithm with parameter p equals to 1%, 5%, 10%, 25%, 50% and 100%. With first four values of parameter the local search algorithm is probabilistic and scans in average 1/100 solutions in neighborhood, 1/20 and so on. If *p*=100% then the algorithm is an usual determined local search algorithm. This experiment has no purpose to define an optimal value of parameter l – the number of algorithm runs, in all cases *l* set equal to 10. Besides that the maximal iterations after each run was set equal to 1000. The algorithm has Borland Delphi 7.0 code. Thorough all experiment Pentium Dual-Core, 2*2,7 GHz was used.

Table 3 contains the results of computing experiment. The average gap with respect to the best known solution was evaluated for 20 benchmarks from each set.

Table 3. LS results with parameter p

№ set	An average gap wrt the best known solution					
p	**0,01**	**0,05**	**0,1**	**0,25**	**0,5**	**1,0**
1	0,6	0,4	0,15	0,15	0,03	0,05
2	1,4	0,65	0,4	0,45	0,4	0,4
3	2,7	2.75	2.65	2.65	2.55	2.55
4	2.65	1.55	1.55	1.6	1.5	1.9
5	1,95	1,0	1,0	0,85	0,45	0,25
6	2,25	1,0	0,95	0,45	0,1	0,05
7	1,35	0,5	0,5	0	0	0
8	0,8	0,35	0,35	0	0	0

As we have 10 runs of algorithms we take the best obtained solution as final solution.

The next Table 4 gives an average solving time (in microseconds) for one benchmark problem from all sets.

Table 4. An average solving time for LS algorithm

№ set	An average time (ms)					
p	**0,01**	**0,05**	**0,1**	**0,25**	**0,5**	**1,0**
1	0,31	1,08	0,89	0,61	0,88	0,81
2	6,38	9,18	12,63	13,08	16,49	15,32
3	37,3	39,8	41,2	38,8	42,5	44,1
4	945	1003	1040	967	1765	2304
5	1,71	1,75	1,49	1,61	1,82	0,81
6	15	19	17	11	12	9
7	38,5	34,3	29,7	35,2	30,1	38,1
8	49,1	51,0	48,0	43,2	47,9	55,6

The results of computing experiment prove that the local search algorithm obtains solutions in many cases close to optimal in a reasonable time. The search performed thorough non-determined neighborhood required less time than the search using full neighborhood. The gap wrt the optimum in many cases slightly worse or the same as the gap when *p=1.0*. The advantage of the determined algorithm in some cases can be explained by fact that searching in probabilistic neighborhood we have a significant deviations of loss or criterion function either best side or worse side. Besides that we have noted that when parameter *p* is small the loss function values have the big gap wrt the best-known solution value. This fact allows making a conclusion that the neighborhood should not be too narrow. So using a small value of parameter p is not effective. By other hand decreasing the number of solutions in neighborhood by quarter (*p=0.25*) or half (*p=0.5*) allows to obtain solutions close to optimum in a less computing time than determined algorithm. In general we can see that the best results were obtained for the bench mark set number 4. It is benchmarks with a big dimension, number of items *n=1000* and with uniformly distributed weights of items. There is no a significant advantages of non-determined algorithm wrt determined for the triplets in sets 5-8. But we can observe a curiosity fact for triplets that the effectiveness of algorithm increases with dimension increasing.

So the proposed local search algorithm can be recommended for solving the problems with great dimension and uniformly distributed weights of items. Besides that this algorithm has a quite small computing time and can be used as a part of more complicated metaheuristics. In particularly author realized the tabu search method using local search both in polynomial and exponential neighborhoods [6,7].

5. Conclusion

1. In this paper Bin Packing Problem was considered and some simple probabilistic heuristics for effective solving it were described.
2. The local search algorithm using as initial solution obtained by simple heuristics was proposed.
3. The definition of two neighborhoods with polynomial power wrt problem dimension was given.
4. The convenient criterion function for solution choosing in neighborhood with constant computing time was offered.

5. The computing experiment results are given. The purpose of experiment was to estimate the effectiveness of the proposed algorithm and to investigate the algorithm behavior at various values of the parameter responsible for the degree of non-determinism of the algorithm.

References

1. Garey M.R. and Johnson D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Frecman, San Francisco. 1979.
2. Martello S.,Toth P. Knapsack Problems, Algorithms and Computer Implemetations. John Wiley and Sons Ltd.- England, 1990.
3. Kochetov Y.A., Usmanova A.R. Probabilistic search with exclusions for the packing problem in containers.- Proceedings of the XII International Conference Baikal, 2001, pp. 22-27.
4. Usmanova A.R. The probabilistic greedy heuristics for the packing problem n containers.- Proceedings of the Optim 2001. First All-Russian scientific-practical conference on solving optimization problems in the industry. St. Petersburg, 2001, pp.141-145.
5. Falkenauer E. A hybrid Grouping Genetic Algorithm for Bin Packing. Journal of Heuristics, Vol.2, No 1,1996.- pp.5-30.
6. Usmanova A.R. Search in exponential neighborhood solutions for linear packing problem. Proceedings of the international conference "Information Technologies for Intelligent Decision Making Support", May 21-25, Ufa, Russia, 2013. Vol.2. pp.189-194
7. Usmanova A.R. Structures of prohibitions list in the method TS for packing task. Proceedings of the international conference "Information Technologies for Intelligent Decision Making Support", May 21-25, Ufa, Russia, 2013. Vol.2. pp. 208-211