VICTORIA UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF COMPUTER AND MATHEMATICAL SCIENCES

An Exchange Heuristic for the Bin Packing Problem

Lutfar R. Khan

(78 OR 5)

July, 1996

(AMS : 90C27, 90-04, 90-08)

TECHNICAL REPORT

AN EXCHANGE HEURISTIC FOR THE BIN PACKING PROBLEM

Lutfar R. Khan

Department of Computer & Mathematical Sciences
Victoria University of Technology, Footscray Campus
PO Box 14428 MCMC
Melbourne 8001
AUSTRALIA

email: khan@matilda.vut.edu.au

Presented at the 14th Triennial Conference of The International Federation of Operations Research Societies (IFORS), Vancouver, British Columbia, Canada

July 8 - 12, 1996

AN EXCHANGE HEURISTIC FOR THE BIN PACKING PROBLEM

Lutfar R. Khan

Department of Computer & Mathematical Sciences
Victoria University of Technology, Footscray Campus
PO Box 14428 MCMC
Melbourne 8001
AUSTRALIA
email: khan@matilda.vut.edu.au

***Abstract*.** For some instances of the bin packing problem, the FFD heuristic performs poorly resulting in a set of near-full bins and one near-empty bin. An algorithm incorporating the mutation operator of genetic algorithms is presented to solve these problems. It starts with an FFD solution. Then one item is selected from the near-empty bin. This selected item and the items from a subset of near-full bins are ordered randomly, and repacked. The process is repeated until the selected item is accommodated in the repacked bins. Experimental investigation suggests that the quality of solutions by an FFD like algorithm can be greatly improved by rearranging the items randomly.

Keywords: bin packing, heuristics, genetic algorithm

1. Introduction.

A Bin Packing Problem (BPP) can be defined as follows:

Given a set of n items and n (possible) bins, with

w_j = weight of item j
and c = capacity of each bin,

assign each item to one bin such that the total weight of the items in each bin does not exceed c and the number of bins used is minimum.

2. Solution of BPP

BPP is an NP-complete problem. It can be formulated as an integer programming problem, but the solution of this integer program would not be efficient. Among the available exact algorithms, the branch and bound method of Martello and Toth (1990), using a reduction procedure, is the most efficient. In this paper, only the simple one-dimensional BPP as defined in Section 1 will be considered.

Excellent heuristic algorithms exist for BPP and its variants. An interesting study on heuristics for BPP is Hall et al (1988), which describes many variants of heuristics and their performance. Among the most efficient heuristic algorithms available in the literature, First Fit Decreasing (FFD) and Best Fit Decreasing (BFD) are best for the simple one-dimensional bin packing problem. It has been shown that *asymptotically* the worst case solutions by either of the above algorithms is not more than 1.22 times the optimal solution (Johnson et al., 1974). Later *absolute* performance ratio for FFD has been shown to be 1.5 (Shimchi-Levi (1993) and Anily et al. (1994)). As with most other heuristic algorithms, the so-called pathological instances can always be constructed to match the worst-case bound, but in general the performance of these algorithms is much more encouraging. An experiment with 900 instances of BPP generated according to the schemes followed by Martello & Toth (1990), shows that in more than 90% of the instances, the FFD gives the optimal solution. These instances will be further discussed in Section 4.

Evidently, FFD performs very well. For some instances, however, FFD fails to produce an optimum solution. Usually, in these instances, the FFD solution consists of a set of near-full bins and one near-empty bin. An example follows.

Example of an instance of BPP:

Bin capacity = 150 Number of items = 20

Ordered set of items weights:

112	95	93	84	80	75	73	73	64	54
52	36	30	30	25	25	23	7	6	2

A Simple Lower Bound is: $\lceil (\Sigma \text{ item weights}) / 150 \rceil = 7$

The FFD Solution is:

Bin No.	i t e m s				Unfilled
1	112	36	2		0
2	95	54			1
3	93	52			5
4	84	64			2
5	80	30	30	7	3
6	75	73			2
7	73	25	25	23	4
8	6				144

This suboptimal FFD solution can be improved, if the item of the eighth bin (weight 6) is selected and repacked with the items of a few other bins. In this instance, the third and seventh bins, which have the largest unfilled capacities, are chosen and the items are repacked to produce the following solution which is optimal.

Bin No.	i t e m s				Unfilled
1	112	36	2		0
2	95	54			1
3	93	25	25	6	1
4	84	64			2
5	80	30	30	7	3
6	75	73			2
7	73	52	23		2

In this instance, an exchange of the items among a few bins, *after* a run of FFD, has been sufficient to produce the optimal solution. While the rearrangement of the items can be designed to be carried out in a deterministic way, a random method can be much faster, albeit without a guarantee of finding an improved rearrangement even though one exits. The idea of random search for improvement is explored in this study in the proposed algorithm.

3. A Heuristic Algorithm

The steps of the proposed algorithm are as follows.

Step 1: Solve the given instance of BPP by FFD. Let the number of required bins be **m**.

Step 2: If m is equal to the lower bound L1 ($= \lceil \sum w_j / c \rceil$), STOP. Otherwise order the bins in a non-ascending order of unfilled capacities. Identify one item I_S from the bin B1 at the head of this list.

Step 3: Let the number of bins selected for repacking be **mm**.

Step 4: Randomly reorder all the items of mm bins selected in Step 3 plus I_s. After reordering, pack them by Next Fit algorithm. If the number of bins required is mm or fewer go to Step 5. Otherwise repeat this repacking a specified number of times (MaxIteration). If no packing is possible to accommodate all selected items using mm bins, let mm = mm + 1 (or any other number if so desired, or even mm unchanged) and go to Step 3. If repeated use of random reordering fails to improve the packing, STOP.

Step 5: Remove item I_s from bin B1 and update the items of other bins. If B1 is empty, set m = m – 1 and go to Step 2. If B1 is not empty, select one item (I_s) from bin B1 and go to Step 3.

This algorithm may be seen as an extension to the FFD heuristic, using random reordering. For ease of description, let it be called FFDR algorithm. As is the case in the given example, there are often a number of instances of BPP for which the FFD algorithm produces a set of well-filled bins, and one bin with a few items and a large remaining available capacity. A comparison of the contents of these FFD-produced bins and those of the bins produced by an exact algorithm often shows that a slight rearrangement of the items of some of the FFD bins can in fact produce an optimal solution. Although a systematic procedure can be applied to identify the required rearrangement by using an exhaustive search, a random technique seems more promising in terms of computational effort. The recent success of the meta-heuristics like Genetic Algorithm in solving problems involving combinatorial search provides an incentive to carry out this study.

Genetic algorithms have been applied to various combinatorial problems including BPP (Reeves, 1993; Falkenauer, 1994). A set of initial solutions are created by some suitable algorithm and stored as members of current population. New solutions are generated from which the better ones replace some older members of the population. This process of reproduction continues until no appreciable improvement appears likely.

Some of the factors that contribute to the success of a genetic algorithm are *reproductive operators (crossover, mutation), the fitness function, encoding and decoding*. Research on genetic algorithms and their applications looks at the suitability of different types of crossover and mutation for combinatorial problems. Falkenauer (1994) has been notably successful in using genetic algorithms for BPP. He has solved some instances of BPP that are very difficult to be solved optimally by any deterministic algorithm. Among the role of the operators for a genetic algorithm for BPP, the most influential factors turn out to be the method of representing the items and groups (encoding and decoding), the definition of the fitness function, and the type of crossover. Mutation, which for BPP is equivalent to the dismantling of the groups of items in one or more bins and subsequent regrouping, is not regarded very useful. Falkenauer (1994) and Reeves (1993) found simple mutation largely ineffective or very disruptive BPP. In this proposed algorithm, the method of dismantling the bins and repacking is similar to the mutation operator. It may be noted that Falkenauer & Delachambre (1992) used the bin with the largest unfilled capacity and a few other randomly chosen bins for dismantling and repacking. The idea of mutation has been used in this study in a very limited form, but the benefits appear substantial compared to the effort. Also, the computer programming task is trivial compared to a full genetic algorithm routine.

4. Experimental Results

For experimental investigation, as mentioned earlier, 900 problem instances have been generated following the scheme used by Martello & Toth (1990). Three classes of randomly generated item sizes have been considered:

Class 1: w_j uniformly random in [1, 100]

Class 2: w_j uniformly random in [20, 100]

Class 3: w_j uniformly random in [50, 100]

For each class, three values of bin capacity c have been considered: $c = 100$, $c = 120$, and $c = 150$. For each pair (class, value of c) and for different values of the number of items n ($n = 50, 100, 120, 500, 1000$), 20 instances have been generated.

FFDR has been coded in FORTRAN and run on selected instances of BPP. The time of computation has not initially been an important consideration and hence not recorded. For the optimal solution, the FORTRAN code available in the book of Martello & Toth (1990) has been used. In Tables 1 and 2, the overall results are summarised. Z (FFD) is the number of bins in an FFD solution.

Table 1. Performance of FFDR Heuristic for Different n

n	No. of instances	Z (FFD) = Z(Opt)*	Z (FFD) ≠ Z(Opt)	FFDR run on	Z (FFDR) = Z(Opt)	Z (FFDR) ≠ Z(Opt)
50	180	169	11	11	6	5
100	180	167	13	13	12	1
200	180	156	24	24	10	14
500	180	165	15	15	11	4
1000	180	166	14	0	0	0
TOTAL	900	823	77	63	39	24

Note: * Z (Opt) represents the solution obtained by the exact algorithm of Martello-Toth; in a few cases (68 out of 900), when the algorithm failed to terminate after 500,000 backtrackings, it was interrupted and the best solution so far taken as Z (Opt); this may distort the results somewhat.

Table 2. Performance of FFDR Heuristic for Different Classes of Problems

Class	No. of instances	Z (FFD) = Z(Opt)*	Z (FFD) ≠ Z(Opt)	FFDR run on	Z (FFDR) = Z(Opt)	Z (FFDR) ≠ Z(Opt)
1	300	275	25	17	10	7
2	300	248	52	46	29	17
3	300	300	0	0	0	0
TOTAL	900	823	77	63	39	24

Some very difficult instances of BPP were suggested by Falkenhauer (1994), where the items are generated randomly in such a manner that all the items can be perfectly packed in a given number of bins, with three items in each bin. According to that scheme twenty problem instances have been generated. FFD produces very poor solutions for these problem instances. An experiment has been done with FFDR on them. Of the 20 instances tried (ten 30-item problems and ten 60-item problems), in each case the FFDR improves the FFD solution. Although an optimal solution has never been found by the limited experiment that has been done, the number of bins required by FFDR algorithm is only one more than the optimum, for each instance. None of the popular deterministic heuristics achieve this result.

5. Conclusion

The performance of FFD for most instances of BPP is very good and often the solutions are acceptable even if they are not proven optimal. The extensions to the FFD algorithm suggested in this study can be useful in several ways.

Firstly, in the design of an exact solution algorithm for BPP, an FFDR solution can serve as a better *upper bound* than an FFD solution, and a possible saving of computation time.

Secondly, the FFDR algorithm can be modified to be used to provide *multiple solutions* for an instance of BPP. A deterministic procedure like FFD will always produce identical solutions to a problem instance. However, if the random reordering is repeated, a set of *different* solutions can be obtained by FFDR. In fact one strong argument for using genetic algorithms is its capacity to provide multiple solutions at any stage of the algorithm, although their quality may not all be acceptable.

The computer code for FFDR is still under development. Initially it has been written in an interactive manner so that the user can conduct the search in a controlled way for experimentation. In the computations described above a maximum of five attempts have been made to eliminate an item from the near-empty bin, and a maximum of 5000 random reorderings has been allowed. In most cases however, when a repacking of the items has been possible, it has been achieved in well within 1000 reorderings. Further experimental study is required to characterize the search in terms of a) the optimal number of bins or items for selection for reordering, b) the determination of whether a relatively large or small item from the near-empty bin should be selected first, and c) the characterization of the problem structures for which this type of algorithm works well. Early indications are that four to six bins should be selected, fewer bins may fail if there are not enough combinations, and more bins will fail if there are too many combinations of items. The influence of the sequence of items selected from the near-empty bins does not seem to have much effect on the final solution. As regards problem structures, problems that are hard to be solved optimally by an exact algorithm (e.g., Class 2 types), are also hard to be improved by FFDR.

As discussed in a recent paper by Dowsland (1995), genetic algorithms are gradually being applied to OR problems, despite having a relatively slow start when compared with simulated annealing and tabu search. She points out that one of the obstacles to its use is the programming task of the algorithm. Indeed the initial set up for a genetic algorithm code is relatively difficult; the idea of mutation used here in a simple setting can demonstrate the benefits of random search and shed some light on the potential of the genetic algorithms as well.

6. References

Anily, S., J. Bramel and D. Shimchi-Levi (1994), "Worse-Case Analysis of Heuristics for the Bin Packing Problem with General Cost Structures", *Operations Research*, Vol. 42(2), pp. 287-298.

Dowsland, K. A. (1996), " Genetic Algorithms -- a Tool for OR?", *Journal of the Operational Research Society*, Vol. 47, pp. 550-561.

Falkenauer, E. and A. Delachambre (1992), "*A Genetic Algorithm for Binpacking and Line Balancing*", Proceedings of 1992 IEEE International Conference on Robotics and Automation (RA92), pp. 1186-1193, Nice 1992.

Falkenauer, E. (1994), "*Setting New Limits in Bin Packing with a Grouping GA Using Reduction*", Technical Report RO108, Department of Industrial Automation, Research Centre for the Belgium Metalworking Industry, Brussels, Belgium, 1994.

Hall, N., S. Ghosh, R. Kankey, S. Narasimhan, and W.-S. Rhee, (1988), " Bin Packing Problems in One Dimension: Heuristic Solutions and Confidence Intervals", *Computers and Operations Research*, Vol. 15(2), pp. 171-177.

Johnson. D., A. Demers, J. Ullman, M. Garey and R. Graham, " Worst Case Performance Bounds for Simple One-Dimensional Packing Algorithms", *SIAM Journal of Computing*, Vol. 3, pp. 299-325.

Martello, S. and P. Toth (1990), "*Knapsack problems: Algorithms and Computer Implementations*", John Wiley & Sons, NY.

Reeves, C. (1993), "*Hybrid Genetic Algorithms for Bin-Packing and Related Problems*", in Proceedings of Annual Conference of Operational Research Society, September, 1993, York.

Shimchi-Levi, D. (1994), "New Worst Case Results for the Bin-Packing Problem", *Naval Research Logistics*, Vol. 41, pp. 579-585.