Evolutionary Heuristics for the Bin Packing Problem

Sami Khuri
Department of Mathematics & Computer Science
San José State University
One Washington Square
San José, CA 95192-0103, U.S.A.

Martin Schütz
Universität Dortmund
Fachbereich Informatik, LS XI
D–44221 Dortmund, Germany

Jörg Heitkötter
EUnet Deutschland GmbH
Research & Development
Emil-Figge-Str. 80
D–44227 Dortmund, Germany

Abstract

In this paper we investigate the use of two evolutionary based heuristic to the bin packing problem. The intractability of this problem is a motivation for the pursuit of heuristics that produce approximate solutions. Unlike other evolutionary based heuristics used with optimization problems, ours do not use domain-specific knowledge and has no specialized genetic operators. It uses a straightforward fitness function to which a graded penalty term is added to penalize infeasible strings. The encoding of the problem makes use of strings that are of integer value. Strings do not represent permutations of the objects as is the case in most approaches to this problem. We use a different representation and give justifications for our choice. Several problem instances are used with a greedy heuristic and the evolutionary based algorithms. We compare the results and conclude with some observations, and suggestions on the use of evolutionary heuristics for combinatorial optimization problems.

Introduction

The bin packing problem (bpp) consists in placing n objects, each of which has a weight ($w_i > 0$), in the minimum number of bins, such that the total weight of the objects in each bin does not exceed the bin's capacity. All bins have the same capacity ($c > 0$). The bpp can be viewed as a multiple subset problem. The goal is to partition n positive numbers (the weights) into the smallest possible number of subsets B_j (the bins), such that the sum of the integers in each subset does not exceed a positive number c.

The bpp belongs to the class of NP-hard problems. The tripartite matching problem can be reduced to it [10]. Thus, unless $P = NP$, the search for exact algorithms is replaced by the design of approximate algorithms. Incidently, only a handful of exact algorithms do exist. Martello and Toth's branch-and-bound based "first-fit decreasing" heuristic is the most adequate algorithm for large problem instances [9].

In this work we describe the results of applying a genetic algorithm (GA) [6, 7] and an Evolution Strategy (ES) [12] to the bpp. The approach used here differs in more than one way to the traditional evolutionary algorithms. First, unlike most encodings that are binary, our population in the GA consists of n-ary strings $x_1 x_2 \dots x_n$, where each x_i, $1 \leq x_i \leq n$, is a real number. The ES makes use of integer valued parameter vectors rather than the standard real-valued parameters. Second, unlike many traditional approaches that use domain-specific knowledge and specialized genetic operators, the work presented in this paper makes use of a graded penalty term incorporated in the fitness function of the GA as well as the ES.

In the next section, we give the formal definition of the bin packing problem including three greedy based heuristics. The paper then shifts to the evolutionary based heuristics. The different ingredients with justifications, for our GA implementation are then presented and encoded in LibGA [3], the GA package we use in this work. For the ES, we make use of the package EVOS$^+$ [11], which was successfully used as an optimization tool for mixed integer problems in the field of optics [1]. In the third section, several problem instances are used with both evolutionary based heuristics and the greedy heuristic and the results are compared. Our work concludes with some observations from our findings, and some suggestions on the use of evolutionary heuristics for other combinatorial optimization problems.

The Bin Packing Problem

The following is a formal definition of the bpp in which we make use of Stinson's terminology for combinatorial optimization problems [13] and in which we introduce concepts and notations that we use in subsequent sections of the study.

Problem Instance:

$$\begin{array}{lcccc} \text{Objects:} & 1 & 2 & \dots & n \\ \text{Weights:} & w_1 & w_2 & \dots & w_n \end{array}$$

n bins, each of capacity c. The weights and capacity are positive real numbers and $w_i < c$ for $i = 1, 2, \dots, n$.

Feasible Solution:

A vector $\vec{x} = x_1 x_2 \dots x_n$ where $x_i = j$ means that the i^{th} object is placed in bin j, $1 \leq j \leq n$; and for every j we have

$$\sum_{i \in B_j} w_i \leq c$$

where $B_j = \{i \mid x_i = j\ \}$; i.e., B_j represents the objects that are in bin j.

Objective Function:

A function $P(\vec{x})$ = number of different values of the components of $\vec{x}$, where $\vec{x} = x_1 x_2 \dots x_n$ is a feasible vector. In other words, $P(\vec{x})$ denotes the number of bins required for the solution represented by $\vec{x}$.

Optimal Solution:

A feasible vector $\vec{x}$ that gives the smallest possible $P(\vec{x})$.

The following are straightforward, simple, greedy based heuristics for the bpp. In the following three algorithms we consider placing an object in previously used bins (non-empty) if such a bin can be found, otherwise we place the object in a new one. In the First Fit Algorithm each element is placed in the first bin that has enough room for it, while in the Best Fit, the object is placed in the best bin that can still contain it. The best bin means the most filled one that still has enough room for the object under consideration.

We can do better than both, the First and the Best Fit Algorithms by saving the objects with small weights for the last steps of the procedure, thus preventing them from taking up space that would be better filled with heavy objects first. More formally, the First Fit Decreasing Algorithm first sorts the weights in decreasing order and then uses the First Fit Algorithm. Note that with the three algorithms, we assume that bins are considered in increasing order of their indices.

We define $W = \sum_{i=1}^{n} w_i$ and by borrowing Garey and Johnson's notation [5], we denote the number of non-empty bins obtained when First Fit Algorithm is applied to problem instance I by FF(I); and by FFD(I) the number of non-empty bins when the First Fit Decreasing Algorithm is used with problem instance I. Similarly, let OPT(I) denote the number of bins obtained by the using an exact algorithm (thus OPT(I) is the optimum packing). It can be shown that [5]:

$$FF(I) \leq \left\lceil \frac{2W}{c} \right\rceil \tag{1}$$

$$FFD(I) = \frac{11}{9} OPT(I) + 4 \tag{2}$$

Since Result 2 shows that the First Fit Decreasing Algorithm always yields solutions that are within 22% of the optimal for large problem instances, we chose to implement it so as to compare its performance with our evolutionary based heuristic.

First Fit Decreasing Algorithm

As mentioned above, the heavy objects are considered before the lighter ones. The actual filling of the bins is thus preceded by the reinitialization of the objects in decreasing order of their weights.

Algorithm First Fit Decreasing
  begin
    reinitialize objects in decreasing order of weights
    for $i = 1, n$ **do**

for $i = 1, m$ **do**
  if $B[j] + w_i \leq c$ **then**
    $B[j] := B[j] + w_i$
end

Evolutionary Based Heuristics

First, recall that GAs work with encodings of the problem rather than the problem itself. We use the encoding described in "Feasible Solution", i.e., our population consists of strings of length n, $x_1 x_2 \ldots x_n$, except that we can sometimes get a better range than $1 \leq x_i \leq n$ for the values of x_i. Inequality 1 gives an upper bound on FF(I). Since we are interested in designing a heuristic that outperforms the First Fit Algorithm on the average, it does make sense to restrict the values of x_i to be no greater that m, where $m = \min\{n, \left\lceil \frac{2W}{c} \right\rceil\}$.

Our initial population is thus generated by producing random strings of length n, $x_1 x_2 \ldots x_n$, where $1 \leq x_i \leq m$.

Next we need to define a fitness function.

The bpp is a highly constrained combinatorial optimization problem. Nevertheless, infeasible strings are not tossed out; they remain in the population but their fitness is weakened by adding a penalty term to the fitness function. Moreover, the penalty is graded since all infeasible strings are not equal. Recall that the number of different values of the components of $\vec{x}$ determines the number of non-empty bins used by $\vec{x}$, which we denoted by $P(\vec{x})$ in "Objective Function".

More precisely, our implementations for the GA and for the ES make use of the following fitness function to be minimized:

$$f(\vec{x}) = P(\vec{x}) + s \cdot \left(m + \sum_{j=1}^{m} \max\left\{ w(B_j) - c,\ 0 \right\} \right)$$

where $s = 0$ when $\vec{x}$ is feasible, $s = 1$ when $\vec{x}$ is infeasible, and $w(B_j)$ is the sum of the weights of the objects in B_j, i.e., $w(B_j) = \sum_{l \in B_j} w_l$

The first term of the fitness function yields the number of bins given by $\vec{x}$. The second term is the penalty and is activated only when $\vec{x}$ is infeasible. The first term, m, of the penalty is an offset term whose sole purpose is to guarantee that infeasible vectors will always yield fitness values that are inferior to the ones obtained from feasible vectors. Moreover, the penalty is highly graded; not only does it take into consideration the number of overfilled bins ($\sum_{j=1}^{m}$ in the second term of $f(\vec{x})$), it also takes into account the distance from feasibility for every bin ($w(B_j) - c$).

We next consider a simple problem instance in which we illustrate the concepts and notations we have introduced.

Example

Consider the following problem instance of the bin packing problem:

Objects:	1	2	3	4	5	6	7	8
Weights:	6	7	5	9	3	4	5	4

Suppose that the capacity of each bin is 12.5.

We first compute $m = \min\{8, \left\lceil \frac{2(W)}{12.5} \right\rceil\}$, where $W = 43$ is the sum of the weights of the objects. Thus, $m = 7$. In other words we consider only 7 (rather than $n = 8$) bins.

Consider the string $\vec{a} = 4\ 5\ 6\ 3\ 5\ 1\ 6\ 1$.

The contents of the 5 bins are $B_1 = \{6, 8\}$ since $x_6 = x_8 = 1$, $B_3 = \{4\}$ since $x_4 = 3$, $B_4 = \{1\}$ since $x_1 = 4$, $B_5 = \{2, 5\}$ since $x_2 = x_5 = 5$, and $B_6 = \{3, 7\}$ since $x_3 = x_4 = 6$. Note that $B_2 = B_7 = \emptyset$.

$\vec{a}$ represents a feasible solution since $w(B_i) \leq c$ for $1 \leq i \leq n$. Consequently, $f(\vec{a}) = P(\vec{a}) = 5$.

On the other hand, $\vec{b} = 6\ 5\ 1\ 3\ 1\ 6\ 3\ 1$ represents an infeasible solution. The contents of the 4 bins are $B_1 = \{3, 5, 8\}$, $B_3 = \{4, 7\}$, $B_5 = \{2\}$ and $B_6 = \{1, 6\}$. The total weight of the objects in B_3 is 14 which exceeds the bin's capacity. Since $\vec{b}$ is infeasible, $f(\vec{b}) = P(\vec{b}) + 7 + (14 - 12.5) = 12.5$.

The First Fit Algorithm applied to this problem instance yields 5 bins: $B_1 = \{1, 3\}$, $B_2 = \{2, 5\}$, $B_3 = \{4\}$, $B_4 = \{6, 7\}$, and $B_4 = \{8\}$.

The Best Fit Algorithm applied to this problem instance yields 4 bins: $B_1 = \{1, 6\}$, $B_2 = \{2, 3\}$, $B_3 = \{4, 5\}$, and $B_4 = \{7, 8\}$.

The First Fit Decreasing Algorithm considers the objects according to their weights (in decreasing order) and then uses 4 bins: $B_1 = \{4, 5\}$, $B_2 = \{2, 3\}$, $B_3 = \{7, 1\}$, and $B_4 = \{6, 8\}$. Note that we might get different fillings of the 4 bins since some of the objects have equal weights.

Experimental Runs

For our experimental runs, we make use of the 40 bin packing problem instances found in Beasley's OR-library [2]. These are divided into two groups. The first (binpack4 from the OR-library) has 20 problems, each consisting of 1000 objects and the bin capacity is 150. The second (binpack8) also has 20 problem instances, each consisting of 501 objects with bin capacity 100.

For the ES, we use a $(15, 100)$-ES, i.e. 15 parents and 100 offspring (see [12] for more details) with standard parameter settings. As for the GA, each run consists of 1000 generations with 500 strings in each. We note that only a very small part (500×10^3) of the search space (e.g. m^{1000} where $m \approx 800$ for binpack4) is explored by the GA. No attempt was made to fine-tune any of the evolutionary heuristics' parameters.

Conclusion

In this work, two evolutionary based heuristic are used as approximation algorithms with a highly constrained combinatorial optimization problem. The solutions are compared to one of the best greedy based procedures, namely the First Fit Decreasing Algorithm. As can be seen from Table 1, the results obtained by the First Fit Decreasing Algorithm are better than the ones obtained by the evolutionary based heuristics. Although it might not be very fair to compare a specialized greedy procedure to general purpose evolutionary heuristics, we do offer these two observations in an attempt to shed some light on our results. First, it is very possible that evolutionary heuristics' representation, and more importantly, stochastic genetic operators are not appropriate for problems such as bin packing that exhibit a "grouping property". In other words, our work would tend to agree with Falkenauer's findings about GAs [4]. He believes that neither standard nor ordering operators are suitable for such problems and offers a hybrid Group Genetic Algorithm as a remedy. Second, we believe that our findings are not conclusive; more work has to be done. After all, similar GA approaches have worked quite well with other combinatorial optimization problems [8].

References

[1] Th. Bäck and M. Schütz. Evolution strategies for mixed-integer optimization of optical multilayer systems. In *Proceedings of the 4th Annual Conference on Evolutionary Programming*, 1995.

[2] J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[3] A. L. Corcoran and R. L. Wainwright. Libga: A user-friendly workbench for order-based genetic algorithm research. In *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, pages 111–117. ACM, ACM Press, February 1993.

[4] E. Falkenaur. A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary Computation*, 2(2):123–144, 1994.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability — A Guide to the Theory of NP–Completeness.* Freemann & Co., San Francisco, CA, 1979.

[6] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison Wesley, Reading, MA, 1989.

[7] J. H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, MI, 1975.

[8] S. Khuri, T. Bäck, and J. Heitkötter. An evolutionary approach to combinatorial optimization problems. In D. Cizmar, editor, *Proceedings of the* 22^{nd} *Annual ACM Computer Science Conference*, pages 66–73. ACM, ACM Press, 1994.

[9] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations.* Wiley, Chichester, West Sussex, England, 1990.

[10] C. H. Papadimitriou. *Computational Complexity.* Addison Wesley, Reading, MA, 1994.

[11] M. Schütz. Eine Evolutionsstrategie für gemischt-ganzzahlige Optimierungsprobleme mit variabler Dimension. Diplomarbeit, Universität Dortmund, Fachbereich Informatik, 1994.

[12] H.-P. Schwefel. *Evolution and Optimum Seeking.* Wiley, New York, 1995.

[13] D. R. Stinson. *An Introduction to the Design and Analysis of Algorithms.* The Charles Babbage Research Center, Winnipeg, Manitoba, Canada, 2nd edition, 1987.

binpack4				*binpack8*			
best	**FFD**	**GA**	**ES**	**best**	**FFD**	**GA**	**ES**
399	403	512	509	167	190	202	211
406	411	530	519	167	191	203	213
411	416	525	526	167	190	204	210
411	416	525	532	167	190	206	216
397	402	513	498	167	191	206	213
399	404	514	506	167	190	203	207
395	399	506	497	167	190	201	209
404	408	514	521	167	189	202	208
399	404	511	512	167	191	202	209
397	404	512	513	167	190	206	210
400	404	516	504	167	190	204	212
401	405	515	508	167	190	202	213
393	398	499	494	167	190	206	209
396	401	509	511	167	190	203	205
394	400	509	504	167	189	207	214
402	408	516	516	167	190	202	209
404	407	525	515	167	189	206	209
404	409	515	524	167	191	203	210
399	403	510	514	167	189	203	206
400	406	512	509	167	191	204	209

Table 1: Runs performed by the greedy heuristic (FFD), and the evolutionary heuristics, a Genetic Algortihm (GA), and an Evolution Strategy (ES) on binpack4 and binpack8 from OR-Lib.