University of Tulsa Technical Report UTULSA-MCS-93-8, May, 1993.

A Heuristic for Improved Genetic Bin Packing*

Arthur L. CORCORAN and Roger L. WAINWRIGHT

Department of Mathematical and Computer Sciences, The University of Tulsa,
600 South College Avenue, Tulsa, OK 74104-3189, USA

Keywords: Combinatorial problems, genetic algorithms, bin packing.

1 Introduction

The bin packing optimization problem packs a set of objects into a set of bins so that the amount of wasted space is minimized. The bin packing problem has many important applications. These include multiprocessor scheduling, resource allocation, and real-world planning, packing, routing, and scheduling optimization problems. The bin packing problem is NP-complete [4]. Since there is therefore little hope in finding an efficient deterministic solution to the bin packing problem, approximation methods have been developed. The advantage of these methods is that they have guaranteed packing performance bounds. A survey of approximation algorithms for bin packing and their respective performance bounds are reported by Garey and Johnson [5] in the one dimensional case, Coffman *et al.* [1] in the two dimensional case, and Li and Cheng [8] in the three dimensional case. In many practical applications of bin packing, a small improvement in packing efficiency can result in great cost savings. For example, when a bin represents a truck, ship or airplane, a large sum of money can be saved by using one less bin. Genetic algorithms have been used on the bin packing problem with great success [2, 3, 9].

A genetic algorithm [6, 7] is an iterative procedure which borrows the ideas of natural selection and 'survival of the fittest' from natural evolution. By simulating natural evolution in this way, a genetic algorithm can easily solve complex problems. Furthermore, by emulating biological selection and reproduction techniques, a genetic algorithm can effectively search the problem domain in a general, representation independent manner. The genetic algorithm maintains a population of chromosomes which represent points in the problem domain. Each chromosome is evaluated by a user defined objective function. The genetic algorithm relys on operators such as reproduction, crossover, and mutation to evolve from one generation to the next. Reproduction ensures the fittest chromosome will survive. Crossover and mutation alter the chromosomes so that new points in the solution space are explored.

2 The Schema Theorem

The *schema theorem* [6] forms the basis for a better understanding of how the genetic algorithm works. A *schema* [7] is a similarity template in which different genetic strings can be compared. Schemata allow similarities to be expressed in a concise and powerful way. Suppose there are m copies of schema H in the population at time t. This is denoted by $m(H,t)$. According to the schema theorem, the expected number of H in the population at time $t+1$, denoted $m(H,t+1)$, is governed by

$$m(H,t+1) = m(H,t)f_r(H)p_s(H)$$

where $f_r(H)$ is the fitness of H relative to the population, and $p_s(H)$ is the probability H survives crossover and mutation.

The schema theorem in its present form allows crossover and mutation to disrupt the growth of highly fit schemata. Consider the beneficial schema, H_b. Suppose H_b is highly fit relative to the current population:

*Research supported by OCAST Grant AR2-004 and Sun Microsystems, Inc.

$f_r(H_b) \geq 1$. Thus, one would expect H_b to grow in the population. The growth of H_b is expressed by the schema theorem:

$$m(H_b, t+1) = m(H_b, t) f_r(H_b) p_s(H_b).$$

When $f_r(H_b)p_s(H_b) < 1$, crossover and mutation are sufficiently disruptive, resulting in negative growth of H_b. Crossover and mutation allow new points in the search space to be explored. However, at the same time they may have a disruptive effect on the exploitation of beneficial schemata. The ultimate goal of the genetic algorithm is for beneficial schemata to grow and harmful or nonbeneficial schemata to decline. Hence, we present the following alternative form of the schema theorem which highlights beneficial schemata:

$$m(H, t+1) = \begin{cases} m(H,t)f_r(H) & \text{if } H \text{ is beneficial} \\ m(H,t)f_r(H)p_s(H) & \text{otherwise} \end{cases}$$

This means that for all beneficial schemata, H_b, the survival probability, $p_s(H_b)$, must be one.

The issue is how to ensure $p_s(H_b) = 1$ for all beneficial schemata. Previous researchers tackled this problem by developing custom crossover functions, rewriting the genetic algorithms, or relying on the chance that p_s is 'good enough' to survive. Clearly, the definition of 'beneficial' schemata is problem specific. Yet, the genetic algorithm provides surprisingly good results with no knowledge of the problem other than the information provided by the objective function. Unfortunately, the objective function does not provide enough information for the genetic algorithm to evaluate the relative merits of genes within the chromosome. Thus the genetic algorithm blindly destroys and rebuilds building blocks in the vain hope that beneficial blocks will remain intact. It is our goal to develop a technique that will overcome the destructive forces of crossover and mutation and combine together fit sequences to form longer, more highly fit sequences. We developed a mechanism for the genetic algorithm to preserve beneficial schemata which requires no modification of the genetic algorithm by the user, and no specific knowledge of the problem by the genetic algorithm. The only way to guarantee the survival of beneficial schemata in genetic algorithms is to identify chromosomes that match each beneficial schema and have a mechanism to preserve them through crossover and mutation.

3 Heuristic for Identifying and Preserving Beneficial Schemata

A *sequence* is defined as a contiguous group of one or more genes within a chromosome. Highly fit sequences correspond to beneficial schemata. Once identified, these sequences can be preserved. The genetic algorithm can then manipulate the remaining genes to provide better results in less time. Our heuristic is composed of two processes: a *sliding window* for identification of sequences, and *reduction* for their preservation.

3.1 Sliding Window

The *sliding window* is a mechanism to identify highly fit sequences. The sliding window is defined by *left* and *right* pointers which correspond to the two end genes in a sequence, as illustrated in Figure 1.

Initially when identifying highly fit sequences the window is positioned on the first gene (*left* and *right* point to the first position). Beginning with the first gene in the chromosome, genes are added to the window on the right or removed from the window on the left as needed to form candidate sequences. A candidate sequence is then evaluated with a problem specific function provided by the user to determine its fitness. If the sequence under investigation is considered *short* of a highly fit sequence according to the problem specific objective function, then genes are added to the right of the window one at a time and the sequence is reevaluated. If the sequence is considered *long* for a highly fit sequence then genes are removed from the left of the window one at a time and the sequence is reevaluated. Sequences which are neither *short* nor *long* are considered highly fit.

Figure 2 illustrates the algorithm for the sliding window. By setting `left` and `right` to 1, the window initially consists of the first gene in the chromosome. In the loop, one of three actions is taken depending on the fitness of the sequence under investigation in the current window. First, if the window is too short, `right` is incremented to widen the window. Second, if the window is too long, `left` is incremented to make the window shorter. That is, the left side of the window is slid over while the right side remains in position. Finally, if the window is neither too short nor too long, a highly fit sequence which extends from `gene[left]` to `gene[right]` has been found. The loop continues until the right side of the window reaches the last gene in the chromosome.

The boundary between 'too short' and 'too long' is problem dependent. For example, in bin packing a 'short' sequence does not fill a bin and a 'long' sequence overflows a bin. In the strictest sense, an exact match to some criteria is required for the sequence to be considered highly fit. This corresponds to preserving only optimal sequences. However, fuzziness can be introduced by allowing a small tolerance when matching the window. This corresponds to the preservation of near optimal sequences.

While the sliding window itself is a linear scan of the chromosome, the evaluation of the sequences are problem dependent and may not be as efficient. In many circumstances, the entire sliding window algorithm may be performed in conjunction with the genetic algorithm's objective function. In this way, duplication of effort could be reduced or eliminated.

3.2 Reduction

Once highly fit sequences have been identified, the chromosome can be manipulated to preserve the highly fit sequences during crossover and mutation. In the case of movable, optimal sequences, *reduction* of the chromosome can take place. That is, optimal sequences can be moved to the front of the chromosome so they can be preserved more easily during crossover. The remaining genes form a chromosome of reduced length. This is also problem dependent. In the case of bin packing, a sequence representing a full bin is a separate entity and is easily movable.

Figure 3a illustrates a chromosome to be reduced. Genes to the left of the *reduction boundary* are optimal sequences resulting from previous reductions. The optimal sequence to the right of the *reduction boundary* is appended to the optimal sequence at the left, and the *reduction boundary* is adjusted resulting in the reduced chromosome shown in Figure 3b.

The intent of reduction is to concatenate all of the optimal sequences together in one place. Hence when an optimal sequence is located it is moved to the front of the chromosome along with the others. This usually means that the entire collection of optimal sequences is highly fit according to the problem dependent criteria. However, it is essential that the problem under investigation have easily identifiable optimal subparts or this technique may not be very useful. In the bin packing problem, suppose a sequence of packages can be found that completely fill a bin. The optimally packed bin can be set aside, and the problem is effectively reduced in size. Almost all planning, routing, layout, scheduling and packing problems have easily identifiable subparts and are candidate problems for the techniques presented in this paper.

There are several advantages of chromosome reduction. It effectively shortens the chromosome and concentrates on a smaller portion of the problem. Thus, more work can be done in the same amount of time. Chromosome reduction isolates the superior sequences so they cannot be disrupted. Another very important advantage is that the traditional crossover operators (with minor modification) can still be used without fear of causing disruption of superior sequences.

A possible disadvantage of chromosome reduction is the possibility of premature convergence which misses the global optimum. For example, consider the one dimensional bin packing problem where the list of objects (3,3,4,6,7,7) are to be packed into bins of size 10. The first three objects in the list can be used to create an optimally filled bin. The remaining three objects each require a bin, resulting in a total of four bins. However, the optimal packing uses three bins. This example illustrates a 'worst case' situation. Fortunately, the genetic algorithm maintains a diverse population of chromosomes to help offset this scenario. It is just as likely for other optimal sequences to be present in the pool as for the worst case. This is why the genetic algorithm generally outperforms deterministic approximation algorithms. However, it does suggest that as the genetic algorithm converges, it may be useful to allow reduced chromosomes to be disrupted to some extent. Even when premature convergence cannot be avoided, reduction should at least improve the bounds over the deterministic strategies, such as next fit, first fit, etc.

4 Experimental Results

We used the genetic algorithm package, LibGA [3], to implement our heuristic on a two dimensional bin packing problem. Coffman *et al.* [1] reports the best deterministic algorithm in this case is *split-fit* (SF), which guarantees performance bounds of 1.5 times optimal. Other algorithms include *first fit decreasing* *height* (FFDH) with bound of 1.7 times optimal and *next fit decreasing height* (NFDH) with bound of 2.0 times optimal. We used *next fit* (NF) as the objective function for the following reasons. Unlike the other methods, it guarantees unique packings for every permutation of the objects. NF is by far the simplest of the algorithms to implement. As an objective function which is called upon repeatedly to evaluate chromosomes, its $O(n)$ efficiency proves more cost effective than the $O(n \log n)$ efficiency of the other methods. It avoids the unnecessary step of presorting the objects by decreasing height as done in NFDH and FFDH, where the presort negates the property of unique permutations.

Each entry in Table 1 depicts the average fitness values obtained from executing the bin packing genetic algorithm ten different times using different random seeds. The pool size was 100 in every case. A steady-state genetic algorithm and a generational genetic algorithm were tested. The steady-state algorithm uses a rank-biased selection similar to that in Genitor [10], with replacement by rank. The generational genetic algorithm uses typical parameters including roulette selection with elitism. The crossover used in both types of genetic algorithm was *asexual*, which is simply a swap of two randomly selected genes in a chromosome. The data sets include several contrived, level-oriented data sets denoted as L25, L75, and L100, consisting of 25, 75, and 100 objects, respectively. These data sets when optimally packed consist of a series of optimally packed levels. Several random data sets, R25, R50, and R100, were generated which consisted of 25, 50, and 100 objects, respectively. The results under 'TRAD' in the table are for the traditional genetic algorithm. The results under 'MOS' are for the traditional genetic algorithm augmented with our heuristic. MOS stands for 'Move Optimal Sequences'. The 'move to the front' has an added benefit in the case of bin packing. This movement ensures that the optimal sequences are now aligned on a level boundary. Most likely the sequence in its original position was not aligned on a level boundary!

Table 1 shows that the new heuristic (MOS) outperformed the traditional genetic algorithm at every opportunity independent of whether a steady-state or generational genetic algorithm was used. For example, for the L25 data set, the steady-state converged at 24.8 for traditional and 22.6 for MOS, and the generational resulted in 22.3 for traditional and 20.3 for MOS. The 20.3 obtained for the generational MOS was 1.015 times the optimal of 20. For the R25 data set, the steady-state converged at 29.8 for traditional and 28.9 for MOS, and the generational resulted in 27.7 for traditional and 26.4 for MOS. The estimated optimal packing heights for the random data sets were computed by dividing the sum total of the areas of the objects by the width of the bin. The 26.4 obtained for the generational MOS was 1.069 times the estimated optimal of 24.7. Notice the generational genetic algorithm outperformed the steady-state in every instance. It has been our experience that this is the case for most order-based problems. It can be best explained by considering the diversity in the population for each model. The generational model selects parents from one pool and places children in another. When enough children have been generated, the children replace the parents. This model tends to converge slowly, since it is difficult for the highly fit members to dominate the population quickly. In the steady-state model, parents and children share the same pool. Consequently, highly fit members can quickly dominate the population and cause the genetic algorithm to converge more quickly. The important point here is not that the steady-state algorithm performed worse, but that the heuristic was able to improve the performance of the genetic algorithm under both models. Note that in all cases, the results for the generational genetic algorithm with MOS were much better than the bounds for even the one dimensional bin packing problem. Also, in all cases, the genetic algorithm's results were obtained within three minutes.

5 Summary

In this paper, the concept of highly fit sequences and how they are ignored and even disrupted by traditional genetic algorithms was presented. The sliding window was introduced, and we described how it could be used to identify highly fit sequences. When these sequences are optimal, the chromosome can be reduced and the sequences preserved by movement to the front of the chromosome. This idea was applied with success to a genetic algorithm for bin packing. Most researchers to date have attempted to minimize the negative aspects of disruption. Our research emphasizes the positive aspect of identifying and manipulating optimal sequences.

Acknowledgements

This research has been supported by OCAST Grant AR2-004. The authors also wish to acknowledge the support of Sun Microsystems, Inc.

References

[1] E. G. Coffman, M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal of Computing*, 9(4):808–826, November 1980.

[2] A. L. Corcoran and R. L. Wainwright. A genetic algorithm for packing in three dimensions. In *Proceedings of the ACM/SIGAPP Symposium on Applied Computing*, pages 1021–1030, Kansas City, Missouri, March 1992.

[3] A. L. Corcoran and R. L. Wainwright. LibGA: A user-friendly workbench for order-based genetic algorithm research. In *Proceedings of the ACM/SIGAPP Symposium on Applied Computing*, pages 111–118, Indianapolis, Indiana, February 1993.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, 1979.

[5] M. R. Garey and D. S. Johnson. Approximation algorithms for bin packing problems: A survey. In G. Ausiello and M. Lucertini, editors, *Analysis and Design of Algorithms in Combinatorial Optimization*, pages 147–172. Springer-Verlag, New York, 1981.

[6] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Massachusetts, 1989.

[7] John H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, Michigan, 1975.

[8] K. Li and K. H. Cheng. On three-dimensional packing. *SIAM Journal of Computing*, 19(5):847–867, October 1990.

[9] D. Smith. Bin packing with adaptive search. In *Proceedings of the First International Conference on Genetic Algorithms*, pages 202–207, Pittsburg, Pennsylvania, July 1988.

[10] D. Whitley and J. Kauth. GENITOR: A different genetic algorithm. In *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, pages 118–130, Denver, Colorado, 1988.

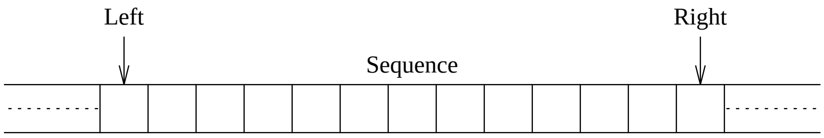

Figure 1: Sliding window

```
procedure Sliding_Window
begin
   left = right = 1;
   while right < chrom_length do
   begin
      evaluate current window
      if window too short then
         right = right + 1;
      else if window too long then
         left = left + 1;
      else
         current window is 'highly fit'
      endif
   end
end.
```

Figure 2: Sliding window algorithm

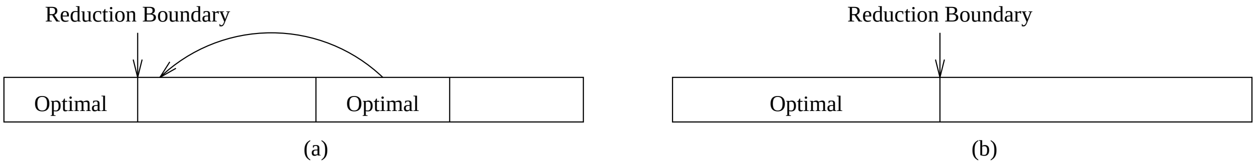

Figure 3: Chromosome reduction

Data Set	Steady-State		Generational		Optimal	Best
	TRAD	MOS	TRAD	MOS		*OPT
L25	24.8	22.6	22.3	20.3	20	1.015
L75	76.5	73.8	69.7	67.7	60	1.128
L100	102.9	101.9	94.5	91.1	80	1.139
R25	29.8	28.9	27.7	26.4	24.7	1.069
R50	56.0	55.2	50.8	49.9	44.8	1.114
R100	106.5	103.9	94.7	94.5	82.6	1.144

Table 1: Bin packing results using TRAD versus MOS