PROCEEDINGS OF THE 1992 ACM/SIGAPP SYMPOSIUM ON APPLIED COMPUTING (SAC'92), Kansas City Convention Center March 1-3, 1992.

A Genetic Algorithm for Packing in Three Dimensions*

Arthur L. Corcoran III
Roger L. Wainwright

Department of Mathematical and Computer Sciences
The University of Tulsa

ABSTRACT

Recent research in Bin Packing has almost exclusively been in two dimensions. In this paper we extend the classic Bin Packing problem to three dimensions. We investigate the solutions for the three dimensional packing problem using first fit and next fit packing strategies with and without genetic algorithms. Five data sets were used to test our algorithms, both random and contrived. They range from 50 to 500 packages. We also studied several existing crossover functions for the genetic algorithm: PMX, Cycle, and Order2. A new crossover function, Rand1, is presented. The genetic algorithm was tested using a randomly generated initial population pool, and a seeded initial pool. The seeded pool was generated from a package ordering produced by rotating and sorting the packages by decreasing height. Our results show the seeded genetic algorithm using Next Fit and PMX produced the best overall results for the data sets tested. The seeded genetic algorithm using Next Fit and Order2 provided the best results considering both rapid execution time and packing efficiently. We found genetic algorithms to be an excellent technique for yielding good solutions for the three dimensional packing problem.

INTRODUCTION

Researchers have recently become interested in solving large combinatorial optimization problems. Finding a solution requires an organized search through the problem space. An unguided search is extremely inefficient since many of these problems are NP-complete. The Bin Packing problem has been shown to be NP-complete. It is one of the classic NP-complete problems. It is impossible to optimally solve any of these problems, except for trivial cases. Consequently, research has focused on approximation techniques which provide efficient, near optimal solutions. Some of these techniques, which would be applicable to bin packing, include heuristic techniques, simulated annealing, neural networks, and genetic algorithms (GA). Papadimitriou and Steiglitz [23] and Parker and Rardin [24] present several classical techniques for solving the bin packing problem. Linear and Dynamic Programming techniques have not been used much in practice since the classic approximation algorithms perform so well on simple problems.

The bin packing problem is applicable in a variety of situations. In computer systems it is used in allocation problems, such as allocating core memory to programs, or space on a disk or tape. Two dimensional bin packing can be used to solve the problem of multiprocessor scheduling with time constraints. The packages represent the time and memory requirements of tasks, and the bins represent processors. The knapsack problem is another closely related problem. In other disciplines, bin packing can be used in such problems as packing trucks, allocating commercials to station-breaks on television, and cutting pipe from standardized lengths. The two dimensional problem can be used for stock cutting, where the packages are patterns which must be cut from a fixed width roll of material (the bin). The work to date on the bin packing problem has been almost exclusively in two dimensions. There has been very little attention given to the three dimensional case.

In this paper we extend the classic Bin Packing problem to three dimensions and investigate various solutions to this problem using genetic algorithms. The rest of the paper is presented as follows. In Section 2 we review the Bin Packing problem. In Section 3 we describe the three dimensional Bin Packing problem. In Section 4 we present an overview of genetic algorithms. Results and conclusions are presented in Section 5. Future research issues and related problems are given in Section 6.

BIN PACKING

In the classic bin packing problem, a finite collection of packages is packed into a set of bins. The packages and bins are characterized by their weights and capacities, respectively. The

* Research partially supported by OCAST Grant AR0-038 and Sun Microsystems, Inc.

problem can be stated either as a decision problem or as an optimization problem. In the decision problem it becomes necessary to determine whether or not there is a disjoint partitioning of the set of packages such that each partition fits into a bin. That is, given an integer number of bins, determine if all of the packages fit into the bins. The optimization problem attempts to minimize the number of bins required, or equivalently, to minimize the amount of wasted bin capacity in the packing.

The bin packing problem is a generalization of the partition problem. That is, the bin represents a partition size and the packages must be optimally placed in these partitions. In most cases, the approximation algorithms (first fit, next fit, best fit, etc.) are very nearly optimal. Unfortunately, there are worst case examples which are far from optimal. These cases only have a few sets of which the packages can belong. Some algorithms (like Modified First Fit Decreasing) have tried to improve absolute bounds by special treatment of these worst cases. However, worst cases can be constructed for the modified algorithms with similar bounds.

The bin packing problem has many parameters which may be varied for different applications. Granularity is one such parameter. The problem could be as small as packing parcels into shipping containers, or as large as packing shipping containers into airplanes. Consequently, a hierarchy of packing problems of differing granularity may need to be solved optimally. The other parameters are variations in packages and bins. See Hu [15] for a more details on bin packing.

The classic bin packing problem is expressed using one dimensional packages. This approach blindly generates partitions so that the sum of the one dimensional package parameters in each partition does not exceed the bin capacity. This parameter is typically stated as the package weight. Clearly, the problem is equivalent if the parameter represents a single spatial dimension. In the real world, a single spatial dimension is often not enough. Problems involving two or three spatial dimensions are typical. The results of one dimensional bin packing may not apply in these cases.

In many problems, packages must arrive at their destinations according to some time constraint. This may be in the form of a deadline ("must arrive before Friday") or a time window ("must arrive between Tuesday and Friday"). The algorithm must ensure the package will not be late. In the case of time windows, it also should not arrive too early. Time constrained scheduling is a research topic in itself, and is a non-trivial additional constraint in bin packing. In addition to individual package attributes, the set of packages themselves may follow a particular distribution. A random distribution might be exhibited in some cases, and a uniform distribution in others. The distribution may be skewed so that packages may only belong to one of a finite number of categories. The package parameters may be nearly equal or relatively small in comparison to the corresponding capacity of the bin. Furthermore, different package parameters could have different distributions.

The classic bin packing problem does not take these parameters into consideration. In fact, the additional parameters make the problem harder. However, by holding all of the additional parameters to some constant value, the harder problems all degenerate back into the classic problem. That is, the classic Bin Packing problem is a special case of the harder problems. For every package attribute there is a corresponding capacity or maximum value for the bin. Generally, there can be no single package with an attribute exceeding the maximum value set by the bin. Otherwise, the package could never be packed. Interesting cases occur when the parameters differ in relation to one another. For example, when the bin is rectangular and the packages are all square.

In the classic problem, the bin has a fixed capacity. For multiple dimensions, the analogous bin would have fixed capacity in every dimension. Thus, a two dimensional bin would define a rectangle, and a three dimensional bin would define a "closed box". A common variation of the classic problem is to use a single, open-ended bin. Used primarily for two or more dimensions, the problem is to minimize the value of the open dimension subject to all other constraints. When using the level technique, this method can be transformed to the classical problem. This is done by packing each level of the single bin into the multiple closed bins, as if each level were a single package. A less common variation uses multiple, dissimilar bins. The distribution of the bins could be like the distribution of package sizes. They could be random, uniform, skewed, etc. It is analogous to packing a fleet of trucks of different sizes and capacities.

One Dimensional Heuristic Techniques for Packing

Many near-optimal heuristic techniques have been developed for the bin packing problem. None of these techniques guarantee an optimal packing. However, many techniques approximate the optimal packing within a constant bound. The three best-known approximation algorithms in bin packing are Next Fit, First Fit, and Best Fit. There are many other algorithms, however, most are variations or refinements of these basic methods and only offer modest improvements in packing efficiency. For more details, see Floyd and Karp [12], Garey and Johnson [13] and Johnson *et al.* [16].

Next Fit is the simplest and easiest algorithm to implement. Beginning with a single bin, the packages are taken from the list in order, and placed in the next available position. When there is not enough room to pack the current package, a new bin is

started. No attempt is made to place packages in previous bins and the packages are considered in the order they appear in the list. Waste can occur since a new bin is used even when a package may fit into a previous bin. This is a linear algorithm in both time and space. Furthermore, it has been shown that Next Fit generates packings no worse than twice optimal. First Fit places each package in the first bin in which it will fit. A new bin is added only when all of the previous bins have been examined and no space can be found for a package. This algorithm can use quadratic time in the worst case and is $O(n \log n)$ on average for n packages. However, the packings produced are no worse than 1.7 times optimal.

Best Fit places each package in the "best" bin in which it will fit. The best bin is the one with the least amount of space left over when the package is added. Surprisingly, this algorithm's asymptotic performance and packing efficiency is identical to that for First Fit. Minor differences in packing efficiency are related to package distribution. That is, for package sizes larger than 1/6 of the bin size (distributed in the range [1/6..1]), Best Fit is more efficient than First Fit. For package sizes larger than 1/5 of the bin size (distributed in the range [1/5..1]) the packing efficiencies are identical. Sorting the packages before applying these methods can lead to improved results. For example, sorting by decreasing package size before applying First Fit results in packings which are no worse than 11/9 times optimal, a 28% improvement. This variation is called First Fit Decreasing. Similar improvements can be found in Next Fit Decreasing and Best Fit Decreasing.

Two Dimensional Heuristic Techniques for Packing

Expanding the problem to two dimensions (rectangle packing) demands different techniques. The first technique uses a "bottom up - left justified" packing rule, or simply "bottom-left". Each package is packed as close to the bottom of the bin and as far to the left of the bin as it can go. This differs from the one dimensional cases where there exists a permutation of the packages for which the methods generate an optimal packing. There are instances where the best bottom-left method produces packings which are 5/4 times optimal. That is, no matter how the packages are ordered, the optimal packing can not be found. The best absolute packing bounds are obtained by sorting the packages by decreasing width. In the special case of uniform square packages, bottom-left degenerates into a one dimensional packing problem. This is because the packages all have the same height, so they are all placed in rows or levels. Since the height is constant, the packing of each level need only be concerned with the package's width. Thus only one dimension is used in the packing.

The "level" technique is a more recent approach that uses this idea. The height of each level is determined by the height of the highest package on that level. A one dimensional (based on width) approach is used to pack each level. For example, Next Fit would be used to pack a level until the next package would not fit. The package would be used to start a new level, and Next Fit would continue as before. Of course, the packages could be sorted by decreasing height or width to improve the result. Surprisingly, the wasted space on each level has no effect on the asymptotic bounds of the problem. Two dimensional First Fit Decreasing Height has the same bound as one dimensional First Fit Decreasing, no worse than 1.7 times optimal.

Classical two dimensional packing generally requires the packing to be orthogonal and disallows rotation of the packages. However, some applications may allow rotation or translation of the packages. Packing efficiency may be improved if each package is rotated so that its width exceeds its height, or vice versa. Several theoretical and practical results are presented in Baker *et al.* [1], Carpenter and Dowsland [3], Coffman *et al.* [5], Dowsland [9] and Leung *et al.* [19].

THREE DIMENSIONAL BIN PACKING

Much of the current research in bin packing has centered on two dimensional strategies. Recall, the most common is the level technique, which places packages level by level into a single, open-ended bin. The packages are presorted by decreasing height and a one dimensional strategy is used to pack the width of the bin. Consequently, two dimensional methods can obtain the same packing efficiency as the corresponding one dimensional methods.

When extending the problem to three dimensions, it is desirable to apply the results of two dimensional research to obtain similar efficiency. Ideally, the packages would be presorted, then placed level by level, using a two dimensional method to pack each level. Unfortunately, it is difficult to extend the packing efficiency in this way. For example, sorting by decreasing height does not guarantee decreasing width or length. Thus, the two dimensional packing may be inefficient. On the other hand, ordering the packages to make the two dimensional packing efficient may cause wasted space to appear in the height. Unlike the purely two dimensional problem, the two dimensional packing stage must deal with a boundary on the second dimension (the length).

Clearly, three dimensional packing is a very practical problem, yet proves to be a very difficult problem to solve. Application of techniques used successfully in one and two dimensional cases fail to yield good solutions. Consequently, little research has been done to date in the area of three dimensional packing. Fenrich *et al.* [10] presents an application of simulated anneal-

ing to three dimensional packing. However, it can only be used on small problem sets.

In an attempt to further understand three dimensional packing, the authors have devised simple three dimensional "next fit" and "first fit" packing algorithms. These algorithms provide a measure with which to compare various package orderings. Thus, the merit of presorting and rotation can be easily evaluated. They also provide an evaluation function useful for various optimization processes. The bin model and algorithm specifics are presented below:

The Bin. Packages are placed in a single, open-ended, three dimensional bin. To avoid the pitfalls of the bottom-left method, a level packing technique is used. Two types of levels are used: horizontal levels along the height of the bin and vertical levels along the length of the bin. To avoid confusion, the levels along the height of the bin will be called "slices". The levels along the length of the bin are analogous to the levels of the two dimensional problem, so will remain "levels". Note, the two dimensional part of the algorithm must deal with a bin whose end is CLOSED.

3D Next Fit. The algorithm begins with a single level on a single slice. A Package is placed in the next available position on the level as long as the width of the bin is not exceeded. In that case, a new level is created and packing continues. The "tallest" (actually, the longest) package on the previous level defines the base of the new level. If at any time the addition of a package would exceed the bin length, a new slice is created. The base of the slice is at the height of the tallest package on the previous slice. The result of the algorithm is the height of the final packing.

3D First Fit. Just as in next fit, packing begins with a single level on a single slice. However, instead of placing the package in the next available position, the levels are searched from the beginning until a place is found in which the package will fit. The width must never exceed the bin width, but may exceed the length of the level as long as the total combined length of all levels does not exceed the bin length. That is, a level can expand to accommodate a package, displacing other levels, if the bin length is not exceeded. Likewise, the height of each slice can expand to accommodate package heights.

GENETIC ALGORITHMS

Goldgerg [14] describes several ways that genetic algorithms differ from traditional algorithms. The genetic algorithm works with a coding of the parameter rather than the actual parameter. The GA works from a population of strings instead of a single point. The genetic algorithm uses probabilistic transition rules, not deterministic rules, and the applications of the genetic operators causes information from the previous generation to be carried over to the next. In addition, genetic algorithms produce "close" to optimal results in a "reasonable" amount of time, and they make no assumptions about the problem space. Furthermore, genetic algorithms are fairly simple to develop and they are suitable for parallel processing.

In a genetic algorithm the parameters of the model to be optimized are encoded into a finite length string, called a chromosome. The fitness of a chromosome determines its ability to survive and reproduce offspring. The genetic algorithm creates an initial population of feasible solutions, and then recombines them in a such way to guide the search to only the most promising areas of the state space. The transition rules that produce one population from another are called genetic recombination operators. These include Reproduction, Crossover and Mutation. Crossover provides new points in the solution space to investigate. Mutation, which occurs rarely, guarantees the entire search space has the opportunity to be searched, given enough time.

Initial population size, evaluation function, crossover method, and mutation rate are parameters which have a great impact on convergence rate as well as the quality of the solution. Larger initial populations help avoid local minima but may make convergence slower. The evaluation function is extremely important for the rate of convergence. This is studied at great length in this paper. If there is little difference between worst case and optimal chromosomes, there will be very little convergence, if any. Larger differences will lead to rapid convergence. Several researchers have investigated the benefits of solving combinatorial optimization problems using genetic algorithms [2,6,7,8,20,21,25,26]. There are several genetic algorithm packages that have been made available to researchers over the past two years or so. One such package is GENITOR, described by Whitney and Kauth [27], which we have installed and modified for our three dimensional Bin Packing problem.

To apply genetic algorithms to the three dimensional packing problem, one must define the encoding of the chromosome, the evaluation function, and the recombination operator. The most natural encoding is to use a string of integers which form an index into the set of packages. The first package is denoted "1", the second "2", and so on. A random reordering of the string represents a random permutation of the packages. The evaluation function returns the height obtained by applying a three dimensional next fit or first fit packing algorithm. The smaller the height, the better the packing.

The recombination operator must produce a permutation of the packages using partial orderings contained in the two parents. The resulting chromosome must include all of the packages with

no duplicates. Fortunately, there are several general purpose crossover functions which meet the requirement. The crossover functions used in our three dimensional packing algorithm include Order2, Cycle, and partially mapped crossover (PMX). These are described by Whitney and Starkweather [28]. In addition, we have developed a fourth technique which randomly selects one of these three methods at each recombination. This method is called Rand1. Provided a positive mutation rate has been specified, mutation will periodically occur during recombination. Mutation swaps a random pair of packages and rotates them about a randomly chosen axis. Mutation is adaptive, that is, the mutation rate increases as the parent chromosomes become more alike.

RESULTS AND CONCLUSIONS

The three dimensional packing algorithm was tested with and without the genetic algorithm. A base set of five data sets was used for comparison. These include:

(1) Random 50, a set of 50 packages with the values for each dimension randomly chosen in the range from 1 to one half the bin size in each dimension.

(2) Random 500, a set of 500 packages with random sizes generated in the same manner as Random 50.

(3) Contrived 320. Figure 1 shows 32 packages of unit height that can be optimally packed. Contrived 320 consists of 10 repetitions of each of the packages shown in Figure 1. The data set is initially placed in no special order. The benefit of this data set is that a 100% fill is possible which may not be true for the random cases.

(4) Contrived 99 is constructed using the packages shown in Figure 2. Figures 2(a) and 2(b) show optimal packings of five and six packages, respectively. The five packages in Figure 2(a) are repeated with heights of one, two and three, yielding a set of 15 packages. The six packages in Figure 2(b) are repeated with heights of one, two, and three, yielding a set of 18 packages. These 33 packages are repeated three times for a total of 99 packages. The Contrived 99 data set is a random ordering of these packages.

(5) Level 75 is constructed using the packages shown in Figure 3. Figures 3(a) and 3(b) show level oriented optimal packings of 15 and 10 packages, respectively. The 15 packages in Figure 3(a) are repeated with heights of one, two and three, yielding a set of 45 packages. The 10 packages in Figure 3(b) are repeated with heights of one, two, and three, yielding a set of 30 packages. The Level 75 data set is a random ordering of these packages.

The optimal packing for Random 50 and Random 500 is not known, thus the optimal fill may not be 100%. However, the optimal packing for Contrived 99, Contrived 320 and Level 75 are 36, 10, and 12 respectively.

The results of the three dimensional packing algorithm without the genetic algorithm are summarized in Table I. The next fit algorithm produced packings of 34.7% and 30.6% fill for the random data sets and between 48% and 50% fill for the contrived data sets. First fit produced 36% fill for the random data sets and from approximately 41% to 53% fill for the contrived data sets. Surprisingly, the next fit and first fit techniques had fairly equal packing efficiencies. Most likely the ability to expand levels to make packages fit reduced the benefits of first fit.

The results of the genetic algorithm for three dimensional packing are summarized in Table II. Both next fit and first fit were used as the evaluation function. For each evaluation function, four different crossover methods were tested. The GENITOR genetic algorithm system adapted for bin packing was used. In all cases, the pool size was 200, the bias was 1.6 and the mutation rate was 0.2. The number of recombinations was 100,000 for all cases except the 320 and 500 package sets, which were run for only 50,000 iterations due to their time requirement.

Running times are not included in Table II, however Order2 was the fastest algorithm. In the Random 50 case using next fit, Order2 took approximately 4.4 minutes to complete while the others took between 6.3 and 7.4 minutes to complete. In the Random 500 case using next fit, Order2 took 13 CPU minutes to complete, while Cycle, PMX, and Rand1 all took between 1.5 and 2.5 hours of CPU time. The use of first fit as an evaluation function increased these times proportionally.

Results shown in Table II indicate that the PMX crossover operator generally had the best packing height. This was independent of the data set tested and independent of the next fit or first fit evaluation functions. However, PMX generally took the most running time. Order2, however, also yielded good packing heights. The significance of this crossover function is its quick running time combined with good results. It often out performed Cycle in both time and packing height. This is illustrated in Figure 4 which plots the relative convergence rates of Order2, Cycle, and PMX for the Random 500 data set. Notice in all three cases the convergence improved rapidly at first and then tapered off with additional recombinations. This is typical of genetic algorithms. In Figure 5, the diversity of the gene pool is illustrated for the Cycle crossover, next fit evaluation function using the Random 500 data set. The best, median, and worst fitness values at each recombination are plotted. Notice the initial random population has a rather wide range. This tightens

up rapidly, until the difference is not noticeable. This is typical in all of the cases we tested.

Comparing results from Table I and to Table II, the genetic algorithm generated better packings for all data sets tested. The best genetic algorithm was able to produce a 54.9% fill for the random 50 case compared to 36% fill without using a GA. The best GA produced a 40.6% fill compared to 36.1% fill in the random 500 case. In the Contrived 99, Contrived 320, and Level 75 data sets the best GAs were respectively, 70.6%, 58.8%, and 66.7% fill compared respectively to 50%, 52.6% and 50% without using a GA. In the best case, this represents a difference of approximately 20% compared to packing without the genetic algorithm.

The effect of preprocessing on next fit and first fit is shown in Table III and Table IV, respectively. Compare the "Presorted by" columns in Table III and the Table I Next Fit data. For each data set listed in the tables, the best case presorting improved the packing utilization from 34.7 to 41.2%, 30.6 to 50.4%, 50 to 58.1%, 47.6 to 71.4%, and 50 to 60%, respectively. That is, a simple presort of the packages by a particular dimension improved next fit's packing efficiency anywhere from 33-50% in the cases tested.

Table III shows the results of the data sets after being rotated so that height > length > width. The packages are then presorted by a particular dimension. Compare the "Rotated and Presorted" columns in Table III and the Table I Next Fit data. For each data set listed in the tables, rotating, then presorting in a particular dimension in the best case improved the packing utilization from 34.7 to 56.9%, 30.6 to 66%, 50 to 60, 47.6 to 71.4%, and 50 to 76.6%, respectively. That is, a rotation and simple presort of the packages by a particular dimension improved next fit's packing efficient anywhere for to 20-64% in the cases tested. The best packings were obtained by presorting by decreasing height, with one exception. The one exception had slightly better results when presorted by decreasing length. Rotation then presorting by decreasing height yielded the best packings in all cases.

In a similar manner the effects of preprocessing on First Fit are shown in Table IV. Compare the "Presorted by" columns in Table IV and the Table I First Fit data. For each data set listed in the tables, the best case presorting improved the packing utilization from 36.0 to 46.5%, 36.1 to 57.3%, 40.9 to 51.4%, 52.6 to 62.5%, and 44.4 to 54.5%, respectively. That is, a simple presort of the packages by a particular dimension also improves first fit's packing efficient in the cases tested.

Table IV shows the results of the data sets after being rotated so that height > length > width. The packages are then presorted by a particular dimension. Compare the "Rotated and Presorted" columns in Table IV and the Table I First Fit data. For each data set listed in the tables, rotating, then presorting in a particular dimension in the best case improved the packing utilization from 36.0 to 53.4%, 36.1 to 62.8%, 40.9 to 70.6%, 52.6 to 76.9%, and 44.4 to 66.7%, respectively. That is, a rotation and simple presort of the packages by a particular dimension also improves first fit's packing efficient.

Clearly, from Tables III and IV it is best to rotate and presort by height regardless if you use next fit or first fit. Furthermore, from results in Table II we note the genetic algorithm did not perform as well as the rotation and presorting by height using either first fit or next fit. This is because the genetic algorithm begins with a random population drawn from a large domain. Higher packing efficiencies can be obtained by seeding the initial population. For example, the packages are rotated then presorted by decreasing height. The resulting list (one chromosome) is placed in the initial pool. The results are summarized in Table V. Notice in every single case the seeded GA yielded the best performance.

Figure 6 illustrates the effects of a seeded genetic algorithm verses an unseeded GA. Figure 7 is a "blow up" of the seeded graph from Figure 6. The unseeded genetic algorithm provides a larger relative improvement over the seeded genetic algorithm. The seeded algorithm starts off at a better approximation and has more work to improve the packing. However, as Figure 7 illustrates, this improvement is not negligible. Figure 7 also emphasizes how the algorithm converges to a local minima, stagnating at that point for a while until the packing is suddenly improved.

In Summary, the seeded GA algorithm using Next Fit and PMX produced the best overall results for the data sets tested. The seeded GA using Next Fit and Order2 provided the best results considering both rapid execution time and packing efficiently.

FUTURE RESEARCH AND RELATED PROBLEMS

The desire to solve large problems naturally leads to the use of parallel techniques. Fenrich *et al.* [11] and Lin *et al.* [22] discuss some recent work in developing parallel algorithms on a hypercube computer for two dimensional bin packing. Each processor uses a serial algorithm to minimize the packing height of a single, open-ended, two dimensional bin. Some packages may be left out of the packing if they fail to cover a percentage of the width of the bin. When the processors are finished, the packings are recursively combined. Any leftover packages are combined and added to the packing. Fenrich *et al.* report very good results, as well as recommendations on which heuristics give the best performance. Our next project is the implementation of a parallel genetic algorithm using HYPERGEN [17] for three dimensional bin packing and related problems.

The Routing Problem is closely related to the packing problem. In real world situations, blind bin packing is not enough. It is desirable that packages with adjacent or equal destinations be packed in the same bin. For example, all packages bound for the same city should be put on the same plane. Given multiple packages, from multiple sources, going to multiple destinations, one must also ensure the package arrives at its destination using the shortest and least expensive route. Thus the problem becomes a combination of bin packing and the traveling salesman problem. A thorough treatment of this topic is described by Christofides *et al.* [4], and Lawler *et al.* [18].

The vehicle routing problem consists of a set of vehicles with time windows and capacity limits. All customers are serviced from a central location. The problem is to minimize the number of vehicles required to deliver all packages within time constraints and vehicle weight constraints. The routing requirement adds an additional constraint to the bin packing problem. Given a package, it can only be placed in the subset of bins which will eventually arrive at the destination. Ideally, it should be placed in the bin which will arrive at the destination in the least time and with the least cost. As in a network flow problem, the bin should be as fully packed as possible for the entire route. At each stop, packages will be added and removed. Packages may be removed from one bin at an intermediate stop and placed on another bin to reach their destination.

ACKNOWLEDGMENTS

This research has been partially supported by OCAST Grant AR0-038. The authors also wish to acknowledge the support of Sun Microsystems, Inc.

REFERENCES

[1] B.S. Baker, E.G. Coffman, R.L. Rivest, "Orthogonal Packings in Two Dimensions," *SIAM Journal of Computing*, 9 (4), pp. 846-855, November 1980.

[2] D.E. Brown, C.L. Huntley and A.R. Spillane, "A Parallel Genetic Heuristic for the Quadratic Assignment Problem", *Proceedings of the Third International Conference on Genetic Algorithms,* Morgan Kaufmann, 1989.

[3] H. Carpenter and W.B. Dowsland, "Practical Considerations of the Pallet-Loading Problem," *Journal of the Operational Research Society*, 36 (6), pp. 489-497, 1985.

[4] N. Christofides, A. Mingozzi, and P. Toth, "The Vehicle Routing Problem" in *Combinatorial Optimization,* John Wiley, New York, 1989, pp. 315-338.

[5] 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), pp. 808-826, November 1980.

[6] L. Davis, ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[7] L. Davis, ed., *Genetic Algorithms and Simulated Annealing,* Morgan Kaufmann Publisher, 1987.

[8] K.A. De Jong and W.M. Spears, "Using Genetic Algorithms to Solve NP- Complete Problems", *Proceedings of the Third International Conference on Genetic Algorithms,* June, 1989, pp. 124-132.

[9] K.A. Dowsland, "An Exact Algorithm for the Pallet Loading Problem," *European Journal of Operational Research*, 31, pp. 78-84, 1987.

[10] R. Fenrich, R. Miller, and Q.F. Stout, "Hypercube Algorithms for some NP-Hard Packing Problems," *Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications*, pp. 769-76, 1989.

[11] R. Fenrich, R. Miller, and Q.F. Stout, "Multi-Tiered Algorithms for 2-Dimensional Bin Packing," *Proceedings of the Fifth Conference on Hypercube Concurrent Computers and Applications*, pp. 58-63, 1990.

[12] S. Floyd and R.M. Karp, "FFD Bin Packing for Item Sizes with Uniform Distributions on [0, 1/2]", *Algorithmica,* 6 (2), pp. 222-239, 1991.

[13] M.R. Garey and D.S. Johnson, "Approximation Algorithms for Bin Packing Problems: A Survey," in Ausiello, G. and M. Lucertini (eds.). Analysis and Design of Algorithms in Combinatorial Optimization. Springer-Verlag: New York, 1981.

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

[15] T.C Hu, "Combinatorial Algorithms", Addision-Wesley, 1982.

[16] D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey, and R.L. Graham, "Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms," *SIAM Journal of Computing,* 3 (4), pp. 299-325, December 1974.

[17] L. Knight and R. Wainwright, "HYPERGEN: A Distributed Genetic Algorithm on a Hypercube", submitted for review.

[18] E.L. Lawler, J.K Lenstra, A.H.G.R. Kan, D.B. Shmoys (eds.). The Traveling Salesman Problem. John Wiley & Sons: New York (1985).

[19] J.Y. Leung, T.W. Tam, C.S. Wong, G.H. Young and F.Y. Chin, "Packing Squares into a Square," *Journal of Parallel and Distributed Computing*, 10, pp. 271-275, 1990.

[20] G.E. Liepins and M.D. Vose, "Deceptiveness and Genetic Algorithm Dynamics, *Foundations of Genetic Algorithms*, G. Rawling, ed., Morgan Kaufmann Publishers, 1991.

[21] G.E. Liepins and M.D. Vose, "Characterizing Crossover in Genetic Algorithms", submitted to *Annals of Mathematics and Artificial Intelligence*

[22] J. Lin, C. Chang, B. Foote, and J.Y. Cheung, "A SMILE for Packing & Pallet Loading in Three Dimensions," to appear.

[23] C.H. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Inc.: Englewood Cliffs, New Jersey (1982).

[24] R.G. Parker and R.L. Rardin. Discrete Optimization. Academic Press, Inc: New York (1988).

[25] G. Rawling, ed., *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, 1991.

[26] D. Whitney, T. Starkweather, and D. Fuquat, "Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator", *Proceedings of the Third International Conference on Genetic Algorithms,* June, 1989.

[27] D. Whitney and J. Kauth, GENITOR: A Different Genetic Algorithm, *Proceedings of the Rocky Mountain Conference on Artificial Intelligence,* Denver, Co., 1988, pp. 118-130.

[28] D. Whitley and T. Starkweather, "GENITOR II: A Distributed Genetic Algorithm, *Journal of Experimental and Theoretical Artificial Intelligence,* 2(1990) 189-214.

Table I: Three Dimensional Packing without Genetic Algorithm

Type	# Pkgs	Next Fit		First Fit	
		Height	Fill	Height	Fill
Random	50	271	34.7%	261	36.0%
Random	500	2879	30.6	2443	36.1
Contrived	99	72	50.0	88	40.9
Contrived	320	21	47.6	19	52.6
Level	75	24	50.0	27	44.4

Table II: Three Dimensional Packing with Genetic Algorithm

Type	# Pkg	Crossover Method	Next Fit		First Fit	
			Height	Fill	Height	Fill
Random	50	Order2	180	52.2%	206	45.6%
		Cycle	207	45.4	206	45.6
		PMX	171	54.9	192	48.9
		Rand1	189	49.7	191	49.2
Random	500	Order2	2253	39.1%	2171	40.5%
		Cycle	2273	38.7	2126	39.8
		PMX	2134	41.3	2194	37.4
		Rand1	2167	40.6	2143	40.3
Contrived	99	Order2	51	70.6%	70	51.4%
		Cycle	52	69.2	68	52.9
		PMX	52	69.2	64	56.2
		Rand1	53	67.9	72	50.0
Contrived	320	Order2	18	55.6%	18	55.6%
		Cycle	18	55.6	18	55.6
		PMX	18	55.6	17	58.8
		Rand1	19	52.6	17	58.8
Level	75	Order2	20	60.0%	22	54.5%
		Cycle	20	60.0	23	52.2
		PMX	18	66.7	22	54.5
		Rand1	21	57.1	21	57.1

Table III: Effect of Preprocessing on Next Fit (without Genetic Algorithm)

Type	# Pkgs	Presorted by			Rotated and Presorted by		
		Width	Length	Height	Width	Length	Height
Random	50	33.2%	41.2%	40.1%	36.0%	48.9%	56.9%
Random	500	35.7	45.2	50.4	43.4	50.2	66.0
Contrived	99	51.4	57.1	58.1	57.1	58.1	60.0
Contrived	320	66.7	71.4	71.4	71.4	71.4	71.4
Level	75	44.4	60.0	60.0	50.0	66.7	70.6

Table IV: Effect of Preprocessing on First Fit (without Genetic Algorithm)

Type	# Pkgs	Presorted by			Rotated & Presorted by		
		Width	Length	Height	Width	Length	Height
Random	50	33.8%	41.6%	46.5%	38.7%	41.6%	53.4%
Random	500	37.4	42.5	57.3	42.1	50.4	62.8
Contrived	99	45.0	51.4	49.3	61.0	63.2	70.6
Contrived	320	58.8	62.5	62.5	76.9	76.9	76.9
Level	75	48.0	54.5	54.5	50.0	60.0	66.7

Table V: Genetic Algorithm Seeded with Best Preprocessed Packing

Type	# Pkg	Crossover Method	Next Fit		First Fit	
			Height	Fill	Height	Fill
Random	50	Order2	150	62.6%	168	55.9%
		Cycle	144	65.2	154	61.0
		PMX	152	61.8	168	55.9
		Rand1	146	64.3	166	56.6
Random	500	Order2	1311	67.2%	1380	63.8%
		Cycle	1312	67.1	1379	63.9
		PMX	1317	66.6	1385	63.6
		Rand1	1316	66.9	1380	63.8
Contrived	99	Order2	51	70.6%	48	75.0%
		Cycle	53	67.9	47	76.6
		PMX	52	69.2	47	76.6
		Rand1	53	67.9	48	75.0
Contrived	320	Order2	14	71.4%	13	76.9%
		Cycle	14	71.4	13	76.9
		PMX	13	76.9	13	76.9
		Rand1	13	76.9	13	76.9
Level	75	Order2	16	75.0%	17	70.6%
		Cycle	16	75.0	17	70.6
		PMX	16	75.0	17	70.6
		Rand1	16	75.0	17	70.6

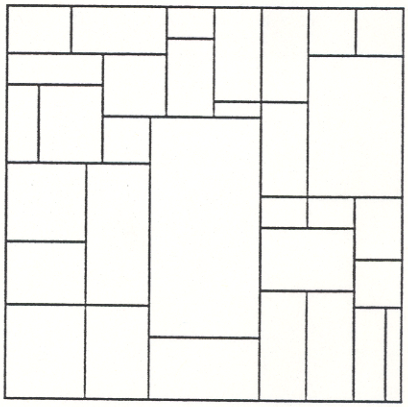

Figure 1: Layout for Contrived 320 Data Set

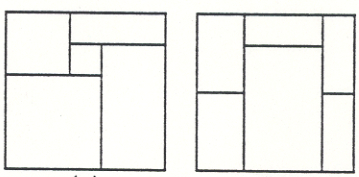

Figure 2: Layout for Contrived 99 Data Set

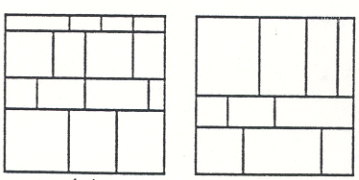

Figure 3: Layout for Level 75 Data Set

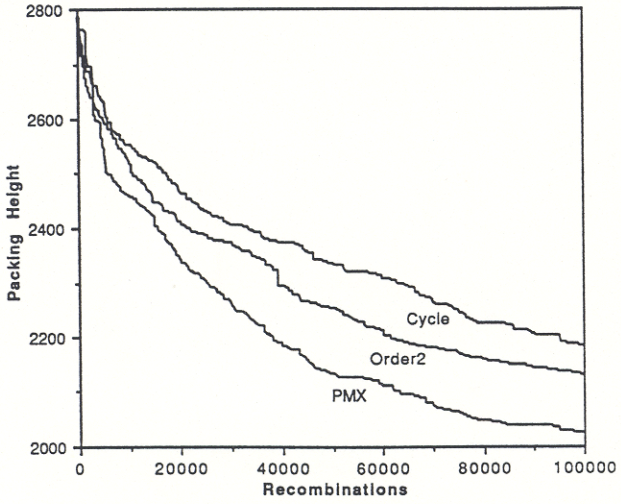

Figure 4: Relative Convergence Rate

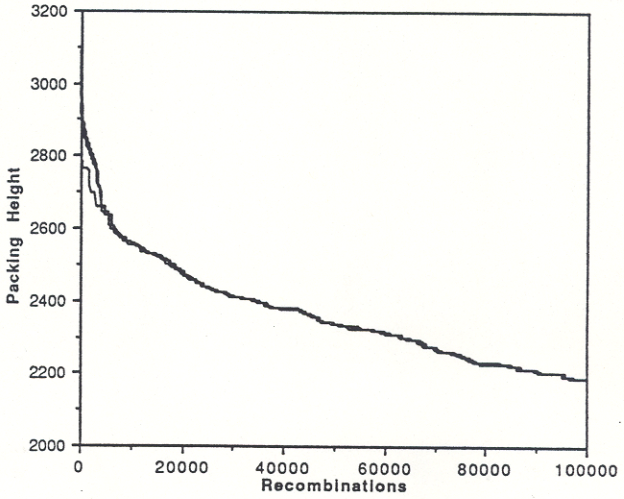

Figure 5: Diversity of the Gene Pool

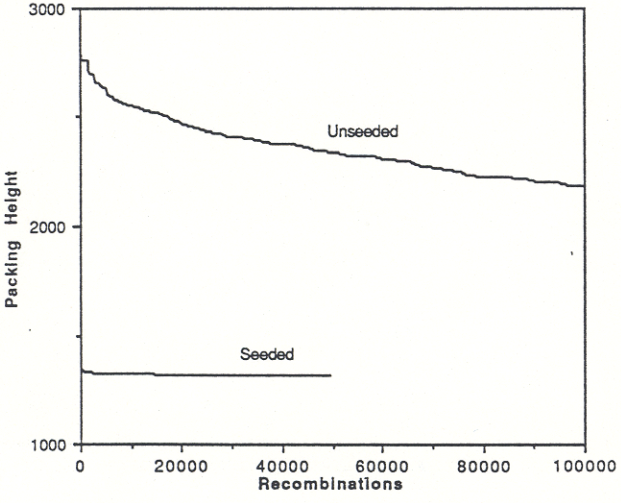

Figure 6: Seeded and Unseeded Genetic Algorithms

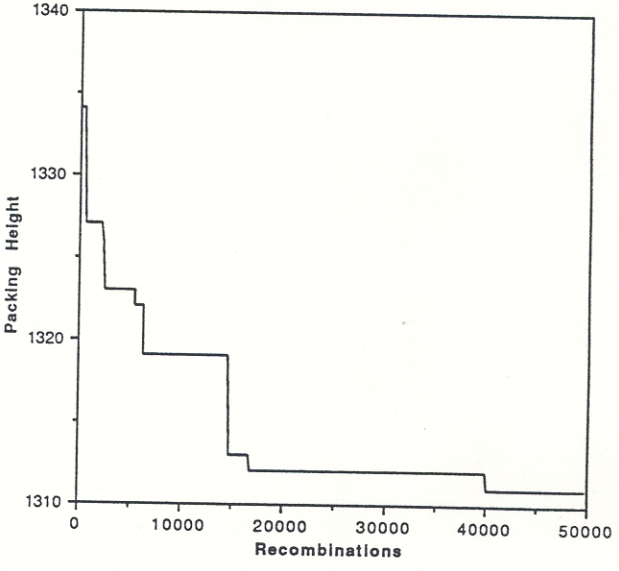

Figure 7: Convergence Rate of Seeded Genetic Algorithm