Solving Virtual Machine Packing with a Reordering Grouping Genetic Algorithm

David Wilcox
CS Deptartment, BYU
Provo, USA
davidw@cs.byu.edu

Andrew McNabb
CS Deptartment, BYU
Provo, USA
a@cs.byu.edu

Kevin Seppi
CS Deptartment, BYU
Provo, USA
k@cs.byu.edu

***Abstract*—We formally define multi-capacity bin packing, a generalization of conventional bin packing, and develop an algorithm called Reordering Grouping Genetic Algorithm (RGGA) to assign VMs to servers. We first test RGGA on conventional bin packing problems and show that it yields excellent results but much more efficiently. We then generate a multi-constraint test set, and demonstrate the effectiveness of RGGA in this context. Lastly, we show the applicability of RGGA in its desired context by using it to develop an assignment of real virtual machines to servers.**

***Index Terms*—Bin Packing, Genetic Algorithm, Virtualization**

I. Introduction

In 2005, data centers in the United States accounted for 1.2% of all U.S. energy consumption, costing $2.7 billion [6]. Part of the problem is that most servers and desktops are in use only 5-15% of the time they are powered on, yet most x86 hardware consumes 60-90% of normal workload power even when idle [14], [3].

Data center costs can be reduced by utilizing VMs (VMs). Using virtualization, multiple operating system instances can be put on the same physical machine in order to more fully exploit hardware capabilities. System administrators can consolidate many VMs on the same physical machine, saving money. To maximize the savings, administrators should pack as many VMs as possible onto a server while satisfying certain performance criteria. We refer to this problem as *Virtual Machine Packing*.

Virtual Machine Packing can be seen as a type of Bin Packing Problem. The Bin Packing Problem is defined as follows: Given a set of items, each with its own weight, find the assignment of items to bins under which the number of bins used is minimized, without violating capacity constraints [1]. However, VM packing is not as simple as the conventional Bin Packing Problem. Packing VMs onto servers is different in that each server has multiple types of constrained resources which the VMs consume. Each VM will add a set amount of load to different resources of the server, such as memory, disk space and CPU. Virtual Machine Packing is an example of one class of bin packing problems where each bin has multiple capacities and each item has multiple weights. The sum of the weights for any given resource must be less than or equal to the corresponding capacity. This is known in the literature as *Multi-Capacity Bin Packing Problem* (MCBPP) [7]. We define it formally in Section II-A. We propose that the Multi-Capacity Bin Packing Problem can potentially be used to model the Virtual Machine Packing Problem.

Bin Packing is NP Hard [1]. Because of the difficulty of the problem, numerous approximation algorithms have been proposed including genetic algorithms (GAs).

We propose a new genetic algorithm to solve the MCBPP. Our new algorithm, the Reordering Grouping Genetic Algorithm (RGGA) is based on multiple representation of individuals in the GA. Multiple representations for each individual in the population are used in order to take advantage of an assortment of genetic operators. This modification allows RGGA to increase production of promising solutions and waste less time producing infeasible solutions.

Our paper is outlined as follows. Section II describes the bin packing problem and heuristics for solving it, including the first fit heuristic. Section II-A describes the MCBPP. Section III then describes how RGGA works with its new combination of genetic operators. In section IV, we will discuss our experimental setup. Finally, section V will discuss and analyze the results of our experiments.

II. Bin Packing

Definition II.1. The *Bin Packing Problem* is formulated as follows. Given a finite set of n items $I = \{1, 2, \ldots, n\}$ with corresponding weights $W = \{w_1, w_2, \ldots, w_n\}$ and a set of identical bins each with capacity $\mathcal{C}$, find the minimum number of bins into which the items can be placed without exceeding the bin capacity $\mathcal{C}$ of any bin. A solution to the bin packing problem is of the form $B = \{b_1, b_2, \ldots, b_m\}$, where each b_i is the set of items assigned to bin i, and is subject to the following constraints:

1) $\forall i \, \exists! \, j$ such that $i \in b_j$ (Every item has to belong to some unique bin.)
2) $\forall j \, \sum_{n \in b_j} w_n \leq \mathcal{C}$ (The sum of the weights of items inside any bin cannot be greater than the bin capacity.)

Definition II.2. An *optimal packing* B to the bin packing problem is one where the particular assignments of items to bins minimizes the number of bins $|B|$.

978-1-4244-7835-4/11/$26.00 ©2011 IEEE

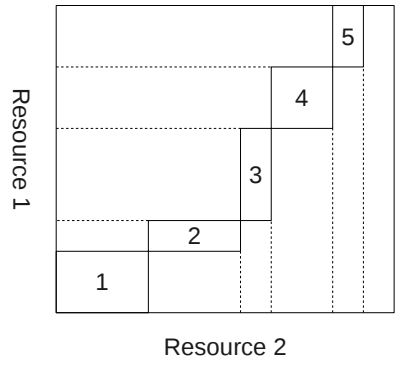

Fig. 1. The two-capacity bin packing problem. The items in each bin are packed regardless of order. Resources used by each item are additive in each dimension.

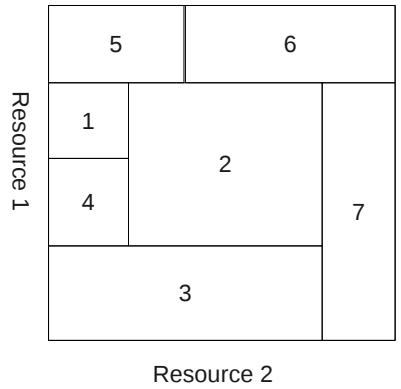

Fig. 2. The conventional two-dimensional bin packing problem. The position and orientation of items in a bin matter.

A. Multi-Capacity Bin Packing

The Multi-Capacity Bin Packing Problem (MCBPP) is different from the conventional multi-dimensional bin packing problem, where the arrangement of items within a bin makes a difference in the end solution. In MCBPP, each dimension is associated with a specific resource; weights are summed independently in each dimension and cannot exceed any resource limit [12], [7]. An example of the two-capacity bin packing problem can be found in Figure 1 in contrast to the two-dimensional bin packing problem in Figure 2.

Definition II.3. The *Multi-Capacity Bin Packing Problem* has the same definition as the conventional bin packing problem in Definition II with some variations. The capacity is a d-dimensional vector $\mathcal{C} = \langle C_1, C_2, \ldots, C_d \rangle$ where d is the number of resources. The weights are redefined so that the weight of item i is a d-dimensional vector $\mathbf{w}_i = \langle w_{i,1}, w_{i,2}, \ldots, w_{i,d} \rangle$.

1) $\forall i \, \exists! \, j$ such that $i \in b_j$ (Every item has to belong to some unique bin.)
2) $\forall j \, \forall k \, \sum_{n_k \in b_j} w_{n,k} \leq C_k$ (The sum of all the weights for any capacity for any bin must be less than the corresponding capacity for that bin.)

B. Solutions to the Bin Packing Problem

There have been numerous approximation algorithms to the conventional bin packing problem. In this section, we will cover the first fit heuristic, genetic algorithms, and other solutions for bin packing.

The first fit algorithm for bin packing is one of the most common heuristics for the problem. The algorithm processes items in arbitrary order. For each item, it attempts to place the item in the first bin that can accommodate the item. If no bin is found, it opens a new bin and puts the item within the new bin. One strength of the first fit heuristic is the running time. It is guaranteed to run in $O(n*log(n))$ where n is the number of items to be packed.

Permutation Pack (PP) attempts to find items in which the largest w components are exactly ordered with respect to the ordering of the corresponding smallest elements in the current bin [7]. For example, if $d = 2$ and $C_1 < C_2$, then we look for an item such that $w_1 < w_2$. If no item is found, the requirements are continually relaxed until one is found. One of the weaknesses of Permutation Pack is the running time. If all permutations are considered, it runs in $O(c!\, n^2)$ where c is the number of capacities and n is the number of items to be packed. We refer the reader to Leinberger et al. [7] for further description of the algorithm.

Genetic Algorithms (GAs) offer another solution to the Bin Packing Problem. There is no rigorous definition for GAs [9]. GAs derive much of their inspiration from Darwinian biological processes. In GAs, individuals represent candidate solutions to the problem. These candidate solutions explore the solution space by undergoing processes similar to those of biological organisms. The simplest form of genetic algorithm involves three types of operators:

- **Selection–**Individuals in the population are selected for crossover with other individuals. Usually, selection is based on elitism, where the more fit individuals are selected more often than less fit individuals.
- **Crossover–**Two individuals in the population exchange subsections of their candidate solution with each other to create two offspring.
- **Mutation–**After crossover, each individual has a probability of having their candidate solution modified slightly (mutated).

These three operators act on the candidate solutions in the population in order to improve the population [9].

Radcliffe et al. [10] proposed six design principles of good representations for individuals in a genetic population. One of the design principles discussed was avoiding redundancy. Avoiding redundancy in the representation allows the GA to search a wider space of possible candidate solutions with fewer computations. Falkenauer [4] discussed that for some types of problems, grouping genetic algorithms (GGAs) avoid redundancy well. In GGAs, the chromosome represents groups. GGAs have been used to solve many types of grouping problems, such as bin packing and graph coloring. One example of a GGA to solve the bin packing problem is Rohlfshagen et

al's [11] exon shuffling genetic algorithm.

Other approaches to the bin packing problem include BISON [2], MBS′, and VNS. The first algorithm, BISON, applies a first fit decreasing strategy to the bins followed by a branch and bound procedure [2]. MBS′ attempts to find a set of items (packing) that fits the bin capacity as much as possible [5]. VNS explores increasingly distant neighborhoods of the current solution and jumps from there to a new one if and only if an improvement has been made [5].

Alvim et al. [1] developed a highly successful heuristic to the bin packing problem which combines multiple algorithms. Their algorithm, HI_BP, uses reduction techniques to eliminate some items and to fix the items in some bins. Afterward, lower bounds and upper bounds are computed. The algorithm halts at this stage if the lower and the upper bounds are equal. Next, a greedy algorithm is applied to build a solution using exactly the number of bins returned by the lower bound (this solution many times will be infeasible). If the solution that was constructed is not feasible, load redistribution strategies are employed. Next, if the solution still is not feasible, a tabu search heuristic is used to attempt to knock down capacity violations. If at this point, the solution is feasible, then the algorithm stops. If the solution is still not feasible, the algorithm adds 1 to the lower bound, and goes back to the reconstruction phase.

As noted by Rohlfshagen et al. [11], Alvim et al.'s approach "relies on several preprocessing steps and undergoes multiple phases exploiting several mathematical properties of the bin packing problem." This approach would not directly extend well to the MCBPP.

III. Reordering Grouping Genetic Algorithm

To solve MCBPP, we propose the Reordering Grouping Genetic Algorithm (RGGA). Because RGGA relies heavily upon the behavior of first fit packing, we prove here that there exists a first fit ordering which will produce an optimal packing for every instance of the bin packing problem. Note that the first fit algorithm as defined works for both the conventional bin packing problem and MCBPP.

Definition III.1. A *packing* B is any feasible solution to a bin packing problem.

Definition III.2. Given a packing B a *packing sequence* $\langle i_{1,1}, i_{1,2}, \ldots, i_{1,n_1}, i_{2,1}, i_{2,2}, \ldots, i_{2,n_2}, \ldots, i_{k,1}, i_{k,2}, \ldots, i_{k,n_k} \rangle$ is a sequence of items formed by a nested iteration over:

- all bins of B in arbitrary order
- and all items within each bin in arbitrary order.

This sequence is not unique—there may be many packing sequences for any given packing.

Definition III.3. A *first fit packing* is obtained by executing the first fit algorithm on a packing sequence.

Theorem III.4. *If B is a packing with $|B|$ bins, and $\mathcal{S}$ is any packing sequence of B with corresponding first fit packing $\mathcal{B}$, then $|\mathcal{B}| \leq |B|$, i.e. $\mathcal{B}$ has no more bins than B.*

Proof: Let $\mathcal{S}_k = \langle i_{1,1}, i_{1,2}, \ldots, i_{1,n_1}, i_{2,1}, i_{2,2}, \ldots, i_{2,n_2}, \ldots, i_{k,1}, i_{k,2}, i_{k,n_k} \rangle$ be the subsequence of $\mathcal{S}$ corresponding to the first k bins. Let $\mathcal{B}_k = \{\beta_1, \beta_2, \ldots \beta_k\}$ be the first fit packing of $\mathcal{S}_k$.

The first fit algorithm will assign $\mathcal{S}_1$ in order into bins in $\mathcal{B}$. Items $i_{1,1}, i_{1,2}, \ldots, i_{1,n}$ will fit into bin β_1 because these items were in a single bin in the packing B. Therefore, $|\mathcal{B}_1| \leq 1$.

By way of induction, suppose that the first fit packing of $\mathcal{S}_k$ uses at most k bins i.e. $|\mathcal{B}_k| \leq k$. We will show that the first fit packing of $\mathcal{S}_{k+1}$ uses at most $k+1$ bins i.e. $|\mathcal{B}_{k+1}| \leq k+1$. The first fit algorithm applied on $\mathcal{S}_{k+1}$ will start by assigning the items in order from $\mathcal{S}_k$ into the first $|B_k|$ bins of $\mathcal{C}$. The remaining items $i_{k+1,1}, i_{k+1,2}, \ldots, i_{k+1,n_{k+1}}$ may also fit into extra space in the first $|B_k|$ bins. In any case, they will require at most one additional bin because they all fit in a single bin, specifically β_{k+1} in the packing B. Therefore, $|\mathcal{B}_{k+1}| \leq |\mathcal{B}_k| + 1 \leq k+1$.

By induction, $|\mathcal{B}| \leq |B|$. ■

Corollary III.5. *For any solvable bin packing problem instance, there exists a packing sequence whose first fit packing is optimal.*

Proof: Let B^* be an optimal packing for the packing sequence, let $\mathcal{S}^*$ be any packing sequence of B^*, and let $\mathcal{B}^*$ be the first fit packing of $\mathcal{S}^*$. By Theorem III.4 $|\mathcal{B}^*| \leq |B^*|$. Because B^* is an optimal packing, $|\mathcal{B}^*| = |B^*|$ and $\mathcal{B}^*$ is optimal. ■

Because there exists some first fit packing for every optimal packing, we can reduce MCBPP to a search for a packing sequence with an optimal first fit packing.

A. Algorithm Overview

Even though Radcliffe et al. [10] proposed that representations of individuals should avoid redundancy, and Falkenauer showed that GGAs represent the problem well, strictly following a grouping genetic algorithm approach is not flexible enough for MCBPP. For example, as will be seen in Section V, the conventional grouping mutation operators do not perform well for MCBPPs. Rohlfshagen et al. [11] proposed a grouping mutation operator that, with equal probability, swaps two items in different bins or moves an item in one bin to another. When this operator is applied to MCBPP with just two capacities, the mutation operator described by Rohlfshagen et al. produced infeasible candidate solutions in over 95% of cases tested. This high percentage of infeasible candidate solutions wastes time and resources when solving the problem. Even though Falkenauer proposed a different, possibly more successful mutation operator, this mutation failed to produce good results when the problem size was large, as we will show in the results. None of the grouping approaches to mutation seem to work well for MCBPP.

RGGA represents individuals in the population not only as an assignment of items to bins, but also as a packing sequence. In order to represent a candidate solution this way, there must be a function to transform a packing sequence to a packing and vice-versa. Using these transformations, we

represent a candidate solution with multiple representations and are able to create operators that take advantage of multiple representations. This gives RGGA the power to use special genetic operators. Not only can RGGA use both ordering and grouping genetic operators, but it can also use operators that have both ordering and grouping components.

When an ordering operator is performed, an assignment of items to bins must be generated. Therefore, the items are first fitted into the solution in order to generate the assignment. When a grouping operator is performed, a packing sequence must be developed. We iterate over each bin and over each item inside of each bin, adding each item encountered to the packing sequence in order.

Because RGGA uses a first fit ordering, it naturally avoids generating infeasible solutions. Other bin packing algorithms need to conduct various searches in order to eliminate infeasible solutions [1]. Because RGGA's solutions are defined as the first fit packing of the list of items, and the first packing will always yield a feasible solution, every solution returned by RGGA will be feasible. This eliminates the time that would be spent cutting down infeasibilities.

B. Fitness Function

Every GA needs a fitness function to evaluate the fitness of an individual in the population. For the conventional bin packing problem, Falkenauer [4] proposed the following cost function of a packing B:

$$f_{BPP}(B) = \frac{1}{|B|}\sum_{j=1}^{|B|} F(b_j) \tag{1}$$

$$F(b_j) = \left(\frac{1}{C}\sum_{i\in b_j} w_i\right)^{\kappa} \tag{2}$$

where w_i is the weight of item i, C the bin capacity, and κ a constant, $1 < \kappa \leq 2$. We use Falkenauer's suggestion of setting $\kappa = 2$.

We adapt this fitness function to MCBPP by incorporating multiple capacities in the fitness function. We first define $F_k(b_j)$ to be the normalized sum of weights if items in bin j and resource k:

$$F_k(b_j) = \left(\frac{1}{C_k}\sum_{i\in b_j} w_{i,k}\right)^{\kappa} \tag{3}$$

where $w_{i,k}$ is the weight of item i with respect to resource k, C_k the bin capacity for resource k, and κ a constant, $1 < \kappa \leq 2$.

We next define $f_{MBPP,j}(B)$ by using equation 1 for one single resource:

$$f_{MBPP,k}(B) = \frac{1}{|B|}\sum_{j=1}^{|B|} F_k(b_j) \tag{4}$$

We define $f_{MBPP}(B)$ by summing $f_{MBPP,k}(B)$ for all resources:

$$f_{MBPP}(B) = \sum_{k=1}^{d} f_{MBPP,k}(B) \tag{5}$$

This function, $f_{MBPP}(B)$, is the fitness function of a single individual B in the genetic population. We must assess whether this metric will always be minimized together with the optimal packing.

Theorem III.6. *Given a multi-capacity bin packing problem, if B^* is an optimal packing and B' is a packing the same items as B^* such that $|B^*| < |B'|$, then $f_{MBPP}(B^*) < f_{MBPP}(B')$.*

Proof: The conventional bin packing problem is a special case of the multi-capacity bin packing problem in that for any fixed k, $f_{MBPP,k}$ is equivalent to f_{BPP}. This, along with a theorem in Falkenauer [4], implies that for any k, $f_{MBPP,k}(B^*) < f_{MBPP,k}(B')$. Therefore,

$$\begin{aligned} f_{MBPP}(B^*) &= \sum_{k=1}^{d} f_{MBPP,k}(B^*) \\ &< \sum_{k=1}^{d} f_{MBPP,k}(B') \\ &= f_{MBPP}(B') \end{aligned}$$

■

C. Crossover Operator

Crossover in genetic algorithms is where individuals in the population combine traits in order to generate a new generation. The child theoretically should have good traits from both parents and hopefully has better fitness.

We choose two parents with probability proportional to the fitness of the individual. In other words, more fit individuals will reproduce with higher probability than less fit individuals.

The crossover operator is implemented using an exon shuffling approach similar to the one described by Rohlfshagen et al. [11]. This approach combines all of the bins from both parents and sorts the bins by fitness. The more full bins are at the front of the list, while the less full bins are at the end. The algorithm systematically picks the more full bins and keeps them intact. If any bin picked contains any item that belongs to a bin that has already been picked, that bin is discarded. This process will produce a list of bins that may not include all items. These remaining items that have not been included in any bin are then sorted in decreasing order and the first fit decreasing heuristic is applied to these items.

Because RGGA uses multiple representations, the resulting assignment of items to bins must be transformed into a packing sequence for other operators. This packing sequence, when first fit packed, may generate a different set of bins. By theorem III.4, we know that the resulting assignment of items to bins after applying the first fit operation will require no more bins than was required before. The only case where the

structure of the items is not preserved is in the case where an item from a later bin is packed into an earlier bin. We consider this a benefit. Because the crossover operator sorts the bins in order of fitness, the earlier bins are generally more tightly packed than the later bins. If the algorithm is able to move any items from later, less tightly packed bins to earlier, tighter bins, the change would be welcome as the later, looser bins now have more space to rearrange.

The operator described here is inherently greedy in that it considers the most tightly packed bin first. Therefore, in order to escape from local optima, we mimic Rohlfshagen et al. [11] by adding noise to the fitness of each bin. The noise of each bin is distributed as a Gaussian with mean $\mu = 0$ and standard deviation $\sigma = \frac{\mathcal{C}}{10}$ where $\mathcal{C}$ is the bin capacity. When bins have multiple capacities, $\sigma = \sum_{i=1}^{|\mathcal{C}|} C_i/10$ where $\mathcal{C}$ is the set of all bin capacities for that bin, and C_i is member i of that set.

D. Mutation Operator

RGGA's mutation operator includes three options. First, Falkenauer proposed a mutation operator where a number of bins are removed from the solution and the items are subsequently reinserted into the candidate solution. Second, two items in the packing sequence of items are swapped. Third, one item is moved to another spot in the packing sequence.

For the second and third genetic operators that work on the packing sequence list, we use information obtained from the groups to improve the performance of the ordering genetic operators. For example, we assure that we never swap two items that came from the same bin. Also, we do not move an item in one bin to the spot of another item from the same bin. Lastly, for all three genetic operators, we mutate items with probability inversely proportional to the fitness of the bin that the item comes from. Items from worse bins are mutated more often than items from better bins. This helps to assure that the structure of better bins is maintained.

We will discuss differences in performance between the three mutation operators in section V. However, for testing RGGA, we settled on applying Falkenauer's remove mutation operator with probability $\frac{1}{3}$, swapping two items with probability $\frac{1}{3}$ and moving an item with probability $\frac{1}{3}$.

IV. Experimental Setup

The GA used for all the experiments was a standard steady-state GA with a population size of 75. The previously discussed crossover operator was applied with probability 0.8. In the event that a crossover is not applied, both parents have a uniform probability of being reproduced asexually. The previously discussed mutation operator was applied with probability 0.1 to each individual after crossover.

Naturally, any new algorithm needs to be compared against existing algorithms. However, because MCBPP described in this paper is new, there are no standard benchmark problems to use for comparison. Therefore, we will test the performance of RGGA in two parts. First, we specify the number of capacities on MCBPP to be one thus reducing it to a conventional bin packing problem. This will allow us to test against conventional evolutionary approaches. Second, we will evaluate the performance of RGGA on a set of MCBPPs generated by us.

A. Conventional Bin Packing Problem

When comparing RGGA to other conventional bin packing algorithms, we will use a set of benchmark instances produced by A. Scholl and R. Klein. The third set, labeled HARD0–HARD9, is a set of triplets. The data set contains 10 problems, each consisting of 200 items with weights between 20,000 and 35,000 and bin capacities of 100,000. We will focus our comparison against other bin packing algorithms using this data set as it is deemed the most difficult. As we will compare RGGA against Exon Shuffling Genetic Algorithm [11], we follow Rohlfshagen et al.'s precedent and execute our algorithm 50 times on each instance, with a maximum limit of 50,000 function evaluations.

B. Multi-Capacity Bin Packing Problem

The goal of this paper is to develop a solution to MCBPP that can be applied to pack VMs onto servers. We developed a set of MCBPPs that models the number of VMs normally found on servers. Based on studies by VMware [13], we generated a test set of MCBPPs that have close to 9-13 items per bin[1]. This test set of multi-capacity bin packing instances is intended to simulate packing VMs onto a cluster of servers. For the purposes of this experiment, we packed each process according to two different resources: processor time and RAM used. We assigned loads to VMs such that nine to thirteen of the VMs can fit on a physical server. The data sets that we used were created such that every item fits perfectly into the set of bins (there is no free space in any bin). We created 10 different two-capacity problem instances of differing size. The smallest problem has approximately 500 VMs while the largest problem has approximately 4500 VMs. An optimal bin packing is extraordinarily difficult to find because each problem was built to have no free space in any of its bins.

Our algorithm was executed 50 times on each problem instance. We initially used a maximum limit of 50,000 function evaluations. However, after analyzing the convergence of the algorithm, we test our algorithm using 7500 function evaluations instead. We report the average number of bins found in the best solution along with the average, minimum and maximum number of function evaluations.

C. RGGA For Virtual Machine Deployments

In order to give validity to this work, we used RGGA to pack real VMs onto servers. We benchmarked RGGA against the algorithms First Fit and Worst Fit. First Fit has been described in Section II-B. The Worst Fit algorithm for bin packing considers each item in order. For each item, it considers only bins that already have at least one item placed in them. It places the item in the bin into which it fits which has the most amount of free space. If the item does not fit in any of the bins considered, it is placed into a new bin.

[1] http://aml.cs.byu.edu/~davidw/rgga/datasets

Approach	Solved
Relaxed MBS' [5]	2
VNS [5]	2
Perturbation MBS' & VNS [5]	2
GA [8]	3
BISON [2], [8]	3
Dual Tabu [2], [5]	3
Exon Shuffling GA [11]	8
RGGA	**9**

TABLE I
OUR ALGORITHM COMPARED WITH OTHER ALGORITHMS ON THE SAME 10 HARD CONVENTIONAL BIN PACKING PROBLEM INSTANCES. THESE INSTANCES ARE NOT REPRESENTATIVE OF MCBPP.

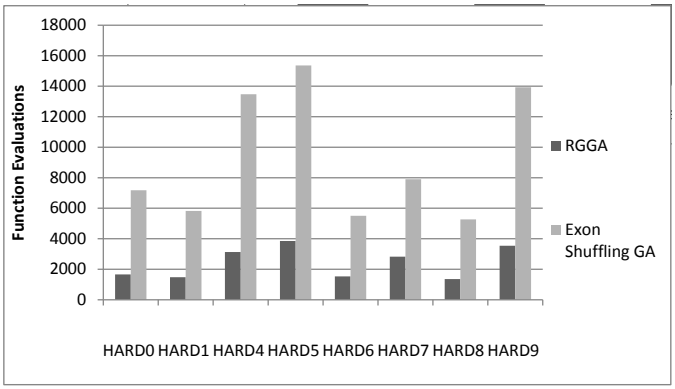

Fig. 3. The average number of function evaluations for the exon shuffling GA are compared with the function evaluations for RGGA.[3]

In our experiments, we created VMs which would have known and deterministic load. We realize that this case does not happen in real deployments, but defer this issue for future research. We created 50 VMs which exhibited a deterministic load on their respective servers. We allowed the packing algorithms to see the loads that these machines exhibit. Then, we used the kvm hypervisor and the qemu monitor to create real VMs on our respective servers. After the VMs had booted, we assigned loads to these VMs and observed resource utilization.

The hardware that we used for benchmarking contain Intel quad core 2.66 gigahertz cpus with cache size of three MB and a RAM size of 8 GB.

V. RESULTS

A. Conventional Bin Packing Problem

We will show that RGGA performs competitively with the best genetic algorithms on single capacity bin packing problems and that it performs well on MCBPPs. Figure I shows a comparison of several approaches in recent literature on the same set of 10 benchmark instances of one-dimensional bin packing.

In most respects, RGGA seems superior. RGGA solves HARD3 to optimality 4% of the time, while exon shuffling genetic algorithm never solves it. For some of the problems that exon shuffling genetic algorithm did not solve completely all the time, such as HARD5, RGGA solves to optimality every time.

[3]HARD2 and HARD3 are ommitted for scaling issues.

[4]HARD0, HARD1, HARD2, HARD6, HARD7 and HARD8 are omitted because both algorithms performed equally well on those problems

instance	Opt	RGGA Avg.	Exon Shuffling Avg
HARD3	55	55.96	56
HARD4	57	57	57.02
HARD5	56	56	56.06
HARD9	56	56	56.02

TABLE II
SUMMARY OF RGGA COMPARED WITH ROLFSHAGEN ET AL.'S EXON SHUFFLING APPROACH.[4]

Opt	RGGA Found	FFD Found	PP Found
59	60.00	61	61.4
112	113.00	121	117.8
191	192.00	207	196.5
216	217.00	234	223.9
241	242.00	262	250.3
267	268.00	296	277.7
320	321.00	353	331.8
371	372.00	412	384.7
425	426.02	465	439.8
481	482.00	541	499.5

TABLE III
SUMMARY OF OUR EXPERIMENTAL RESULTS OF OUR ALGORITHM ON THE MULTI-CAPACITY PROBLEMS.

Even though RGGA performed slightly better in terms of percent of problems solved, it performed much better in terms of function evaluations. Figure 3 shows the average number of function evaluations for RGGA and exon shuffling genetic algorithm for different problems. RGGA performs significantly better in this regard.

B. Multi-Capacity Bin Packing Problem

For the multi-capacity problem, we tested RGGA using the multi-capacity data set described in Section IV. RGGA is meant to solve MCBPPs like this data set rather than instances of the conventional single-capacity bin packing problem. The items in this data set represent VMs and the bins in this data set represent servers. These problems were meant to be very hard as the optimal packing will have no free space in any bin. Figure III shows that RGGA was always one bin away from the optimal packing in every run, except for one run when the optimal number of bins was 425.

Also, Table III shows a comparison of RGGA, Permutation Pack (PP) and the first fit decreasing (FFD) heuristic for each problem instance. Because MCBPP has multiple weights for each item, in order to compare two different items in MCBPP, we use the following scoring, $S(i) = \sum_{j=1}^{k} w_{i,j}^2$ where i is a particular item, $S(i)$ is the score of the item, k is the number of weights belonging to item i, and $w_{i,j}$ is weight j of item i. RGGA significantly outperforms the first fit decreasing heuristic.

In order to show that a grouping representation is not desired all of the time for all genetic operators, we show that the ordering mutations outperform the grouping mutations

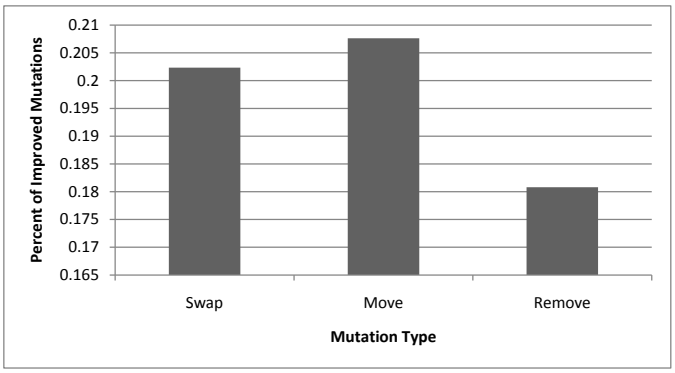

Fig. 4. The average improved solutions for each mutation type for the multi-capacity problems.

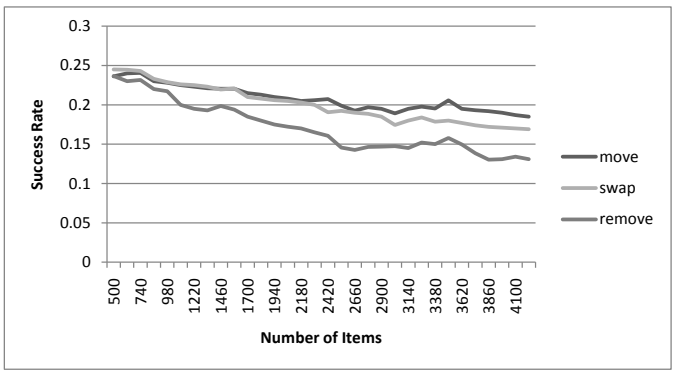

Fig. 5. The average percent of mutations that improved individuals plotted against the number of items in each of MCBPPs.

for large MCBPPs (the type of problems that this algorithm would likely encounter when packing VMs onto servers). In order to show that RGGA's ordering approach to the mutation operator is an improvement upon Falkenauer's grouping approach to mutation for large multi-capacity problems, we compared the percentage of mutations that led to improved candidate solutions for the three different types of mutations that RGGA used. Recall that RGGA used three different mutation operators–remove a few bins and reinsert their contents (remove), move an item in the ordered item list (move), and swap two items in the item list (swap).

We generated a new test set of multi-capacity bin packing instances of corresponding to cases with the number of items between 500 and 4500. We performed this test for problems of varying numbers of items. We show a summary of our findings in Figure 4 for MCBPPs. The graph shows that the move mutation outperformed Falkenauer's remove mutation. Move and swap mutations improved their solutions in about 20% of cases. Falkenauer's remove mutation improved its solutions in about 16% of cases. A more detailed graph of how each operator did on each specific problem size can be found in Figure 5. As can be seen in Figure 5, all of the mutation operators performed nearly equally well for small data sets. However, when the size of the data set increases, the performance of Falkenauer's remove mutation operator diminishes while the move and swap mutation operators still perform fairly well.

A comparison of each mutation's performance for single-capacity problems can be found in Figure 6. For single-capacity problems (HARD0–HARD9), the move and swap mutations did not perform as well in comparison with the multi-capacity problem. The swap mutation performed worse, but did not perform substantially worse than the move mutation and Falkenauer's mutation. It may be that these test problems were small enough that Falkenauer's mutation still performed reasonably well.

C. RGGA For Virtual Machine Deployments

After creating artificial loads on VMs, we observed the total amount of energy used by all servers used. We show in Figure 7 the energy used by the deployments created by the three different algorithms. The deployment created by RGGA uses much less energy than the other benchmarks we used. If loads on virtual machines are deterministic and known, RGGA can take advantage of this, and pack VMs tightly onto servers.

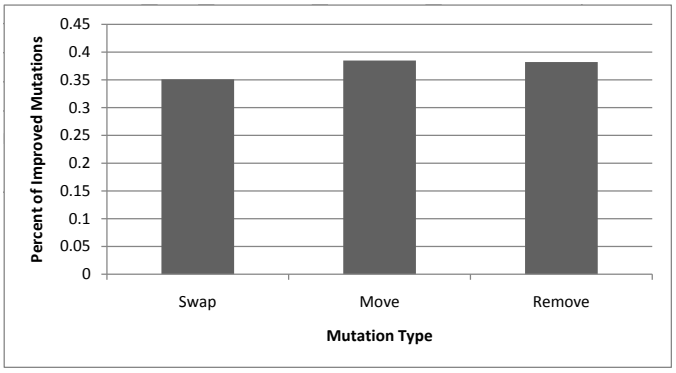

Fig. 6. The average percent of mutations that improved individuals for single capacity problems.

D. Time to Run RGGA

In the Sections V-A and V-B we showed that RGGA is very good at finding good answers. We show here that RGGA also converges very quickly, thus limiting the running time needed when it is used in data centers. We show in Figure 8 the

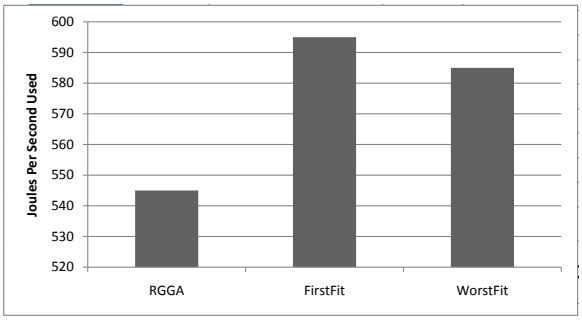

Fig. 7. The amount of energy used in solutions returned by the three different algorithms.

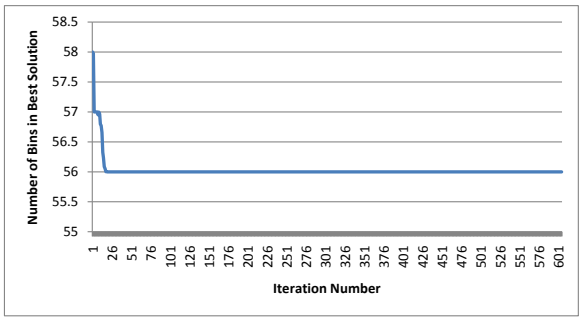

Fig. 8. The average number of bins in the best solution for different iteration numbers for HARD0.

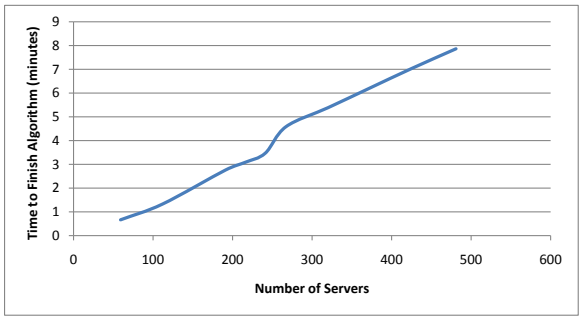

Fig. 9. The amount of time taken to find a solution for varying sizes of Multi-Capacity Bin Packing Problems.

average number of bins in the population at varying iteration numbers when running RGGA on HARD0. As can be seen, RGGA finds an optimal solution in fewer than thirty iterations.

We wished to find out how much real time RGGA would take to run in a real-life situation, as one weakness of GAs is the speed it takes to run them. We imagine that there may be problem sets which may be harder to optimize than these problems here, and thus give RGGA a maximum of 100 iterations or 7500 function evaluations instead of the thirty shown for this problem. We benchmarked the time RGGA needed to find come to a solution after 100 iterations for varying sizes of Multi-Capacity Bin Packing Problems.

We show in Figure 9 the amount of time needed to solve an answer on our hardware. The figure shows RGGA scales linearly within this range as the problem size increases. Moreover, data centers of even 500 servers can be solved in fewer than 10 minutes on a single core. We imagine that most data centers with 500 servers could spare a single core for ten minutes without much trouble in order to optimize their virtual machines more efficiently. This shows that the running time of RGGA is generally feasible to run in data centers.

VI. Conclusions

This paper assumes that workloads for VMs are deterministic. In realistic data center operations, some jobs are batch-scheduled, but others have nondeterministic loads. At first glance, one may conclude that the results of this paper do not apply to these jobs with nondeterministic loads. However, some changes on this algorithm and model could in fact extend this algorithm for nondeterministic work loads.

One possible change to extend this algorithm to nondeterministic loads would be to take varying expectations on the load distributions of VMs. Using these different expectations, instead of merely the mean of each load distribution, it is possible to achieve any arbitrary guarantee of performance. This idea is implemented and explained further in [15].

As well, if it is possible to relocate jobs, then the issue of deterministic becomes less important. It is possible to spend a few minutes every couple of hours getting a better packing. This paper helps to achieve a good initial packing so that relocation would not need to happen as often.

References

[1] F. Glover A. C. F. Alvim, C. C. Ribeiro and D. J. Aloise. A hybrid improvment heuristic for the one-dimensional bin packing problem. *Journal of Heuristics*, 10:4–27, 2004.

[2] R. Klein A. Scholl and C. J. Bison. A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers and Operations Research*, 24(7):627–645, 1997.

[3] Michael Bailey, Timothy Rostrom, and J. Ekstrom. Operating system power dependencies. In *Int. CMG Conference*, pages 15–20, 2008.

[4] E. Faulkenaeur. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

[5] K. Fleszar and K. S. Hindi. New heuristics for one-dimensional bin packing. *Computers and Operations Research*, 29:821–839, 2002.

[6] J. G. Koomey. Estimating total power consumption by servers in the u.s. and the world. *Presented at the EPA stakeholder workshop at Santa Clara Convention Center*, February 2007.

[7] William Leinberger, George Karypis, and Vipin Kumar. Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In *ICPP '99: Proceedings of the 1999 International Conference on Parallel Processing*, page 404, Washington, DC, USA, 1999. IEEE Computer Society.

[8] H. Lima and T. Yakawa. A new design of genetic algorithm for bin packing. *Evolutionary Computation, 2003. The 2003 Congresson Evolutionary Computation*, 2:1044–1049, 2003.

[9] M. Mitchell. *An introduction to genetic algorithms*. MIT Press, 1998.

[10] N. J. Radcliffe. Forma analysis and random respectful recombination. *Proceedings of the Fourth International Conference on Genetic Algorithms*, July 1991.

[11] P. Rohlfshagen and J. Bullinaria. A genetic algorithm with exon shuffling crossover for hard bin packing problems. *Proceedings of Genetic And Evolutionary Computation Conference*, 9:1365–1371, 2007.

[12] Mark Stillwell, David Schanzenbach, Frederic Vivien, and Henri Casanova. Resource allocation using virtual clusters. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 260–267, Washington, DC, USA, 2009. IEEE Computer Society.

[13] VMWare. Server consolidation overview, building a virtual infrastructure, September 2009.

[14] VMware. Vmware green it energy efficiency, reduce energy costs with virtualization, September 2009.

[15] David Wilcox, Andrew McNabb, Kevin Seppi, and Kelly Flanagan. Probabilistic virtual machine assignment. In *In Proceedings of CLOUD COMPUTING 2010, The First International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 54–60, 2010.