Comparing Solutions to the Two-Dimensional Package Placement Problem

Joe Shymanski

April 24, 2023

1 Problem Description

The two-dimensional package placement problem deals with placing a number of packages within a grid in some optimal fashion. More specifically, given grid dimensions and a set of weights (one for each package pair), minimize the sum of the weighted distances between all package pairs. Thus, it is best to place package pairs with large weights closer together than those with small weights.

Several optimization problems fit this mold. Consider circuit components as the packages, the motherboard as the grid space, and the number of wires connecting two components as the weight for a package pair. The goal is to minimize the number of wire crossovers that occur by placing highly-connected components adjacent to each other. Alternatively, a city may wish to reduce the number of pedestrian street crossings by constructing building pairs with heavy foot traffic near one another. This problem clearly generalizes to a large range of applications.

This problem is solved by specifying a permutation of the packages within the grid. Since there are no other constraints on how the packages can be arranged, the search space for this problem is $P!$, where P is the number of packages. This leads to the NP-completeness of the two-dimensional package placement problem, but will not be formally proven here.

a	d	g	a	b	c	c	b	a	g	d	a
b	e	h	d	e	f	f	e	d	h	e	b
c	f	i	g	h	i	i	h	g	i	f	c
c	f	i	g	h	i	i	h	g	i	f	c
b	e	h	d	e	f	f	e	d	h	e	b
a	d	g	a	b	c	c	b	a	g	d	a

Table 1: Eight optimal configurations for square grids.

a	b	c	d	d	c	b	a
e	f	g	h	h	g	f	e
i	j	k	l	l	k	j	i
i	j	k	l	l	k	j	i
e	f	g	h	h	g	f	e
a	b	c	d	d	c	b	a

Table 2: Four optimal configurations for lopsided, non-linear grids.

2 Solution Setup

This paper will test the effectiveness of "foolish" hill climbing, simulated annealing, and a genetic algorithm in generating solutions for this problem. The hill climbing that this paper utilizes is foolish by never taking any backward steps. Since this is a permutation problem, the solution state (for hill climbing and simulated annealing) or chromosome structure (for the genetic algorithm) is simply an ordering of the package IDs, represented by a list of P labels. For example, a chromosome of `[a,d,g,b,e,h,c,f,i]` with dimensions `[3, 3]` corresponds to the upper left grid of Table 1.

For the purposes of testing, the optimal solution for each grid size consists of an alphabetical ordering of the package labels. In addition, an interesting observation about this problem is that there are always multiple, equally optimal solutions. Square grids have a total of eight different optimal configurations (see Table 1) while lopsided, non-linear grids have four (see Table 2) and linear grids have two (see Table 3). One might then assume that these symmetries make it four times as difficult to find solutions to linear grids as opposed to square ones. This will be investigated later.

a	b	c	d	d	c	b	a

Table 3: Two optimal configurations for linear grids.

3 Algorithm Structure

3.1 Genetic Algorithm

The genetic algorithm tested in this paper is generational, not steady-state. It accepts as input population size, maximum number of generations, and maximum amount of time before termination (in seconds); each test run used values of 1000, 100, and 300, respectively. The chromosome representation, as mentioned above, is a permuted list of package IDs. The initial population is generated completely randomly, and each generation retains two elites. If any of the termination criteria are met, or if the algorithm finds an optimal solution, the process ends and returns the best individual in the final population.

The fitness function used for this problem takes in a weight matrix for the package pairs and the dimensions of the grid. It then calculates the product of the weight and Manhattan distance for each package pair. Finally, it outputs the sum of these products. For the purposes of this problem, the algorithm seeks to minimize this output.

The three selection methods tested were roulette (RU), rank (RK), and "diff" (DF). The first two operate as traditionally explained in the literature. The third is a kind of normalized roulette; each fitness score in the population is subtracted from the population maximum plus one. Since the fitness scores in this problem can be in the thousands, it may better highlight the differences between the scores (hence the name).

The two crossover methods tested were cut and crossfill (CC) and order 1 (O1). The cut and crossfill method chooses a random split point in the parent chromosome and randomly chooses either the left or right segment to retain in the child chromosome. It then fills in the remaining portion of the chromosome using the order that the missing genes appear in the other parent. Order 1 chooses two random split points in the parent and copies the segment between the splits into the child chromosome. Then, starting after the second split point, it fills the remaining segment with the missing genes in the order they appear in the other parent (again, after the second split point). The crossover rate is set at 100%.

The three mutation methods tested were single move (SM), pairwise exchange (PE), and cycle of three (C3). Single move chooses a random gene and a random insert point. It then removes the gene from its position in the chromosome and replaces it at the insert location, shifting all the genes in between forward or backward accordingly. Pairwise exchange picks two random genes and swaps their positions. Cycle of three picks three random genes and swaps them in a forward cycle (the first to the second position, second to the third, and third to the first). The mutation rate varies dynamically, ranging on a linear scale from 1% to 10% as the number of generations increases.

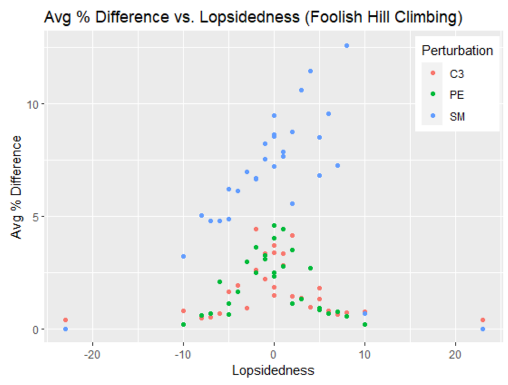
3.2 Simulated Annealing

The algorithm used for simulated annealing in this paper takes in values for the initial temperature (t_0), the initial number of iterations per loop (i_0), the rate of decay in temperature (α), and the rate of growth for the number of iterations per loop (β). For this paper, a loop refers to a set of uninterrupted iterations. All test runs used values of 100, 100, .9, and 1.01, respectively. It also accepts values for two of its termination criteria: maximum execution time (in seconds), the maximum number of consecutive loops in which the heuristic value stays the same, which were set to 300 and 50, respectively.

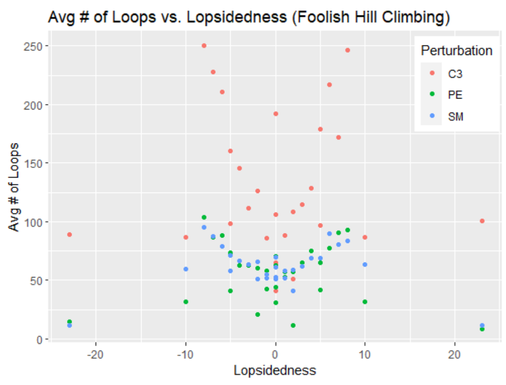
The solution state is identical to the chromosome representation from the genetic algorithm. It is also generated completely randomly at the start. Additionally, the heuristic function used is the same as the fitness function in the genetic algorithm, and the three perturbation functions tested are the same as the mutations.

3.3 Foolish Hill Climbing

The hill climbing algorithm in this paper is exactly the same as simulated annealing, except for one key difference: it never accepts a worse solution than the current one. Thus, it does not utilize the temperature mechanic of simulated annealing, so t_0 and α go unused.



(a) Average percent difference.



(b) Average number of loops.

Figure 1: Lopsidedness analysis for foolish hill climbing.

4 Results and Analysis

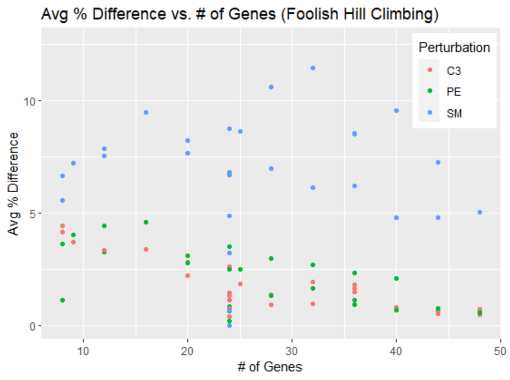
Each data point is the result of averaging each metric across five test runs with the same algorithm settings. For the purpose of analysis, the term "lopsidedness" refers to the number of grid rows minus the number of grid columns. In total, 30 different grid dimensions were tested, and each test on a lopsided grid is complemented by its symmetrical counterpart. The number of packages ranged from 8 to 48, and the optimal fitness ranged from 144 to 48464. The largest search space featured about 10^{61} possible permutations.

4.1 Foolish Hill Climbing

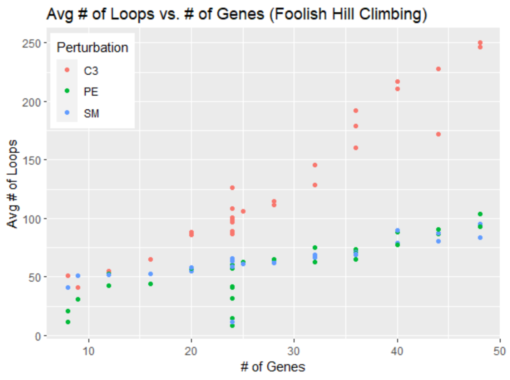
Figure 1 shows the effect of lopsidedness on the performance of foolish hill climbing. Namely, Figure 1a shows that SM provides better solutions as the grid becomes wider and shorter.

Figure 2 shows the effect of the number of genes (packages) on the performance of foolish hill climbing. As the problem size grows, both the accuracy and the number of loops also increase.

Finally, Figure 3 shows the distinction between the performance of the three perturbation functions. C3 takes many loops to converge to a fairly accurate solution, whereas SM stalls out quickly at a poor answer. PE appears to be the best of both worlds, finding accurate permutation in a small number of loops.



(a) Average percent difference.



(b) Average number of loops.

Figure 2: Problem size analysis for foolish hill climbing.

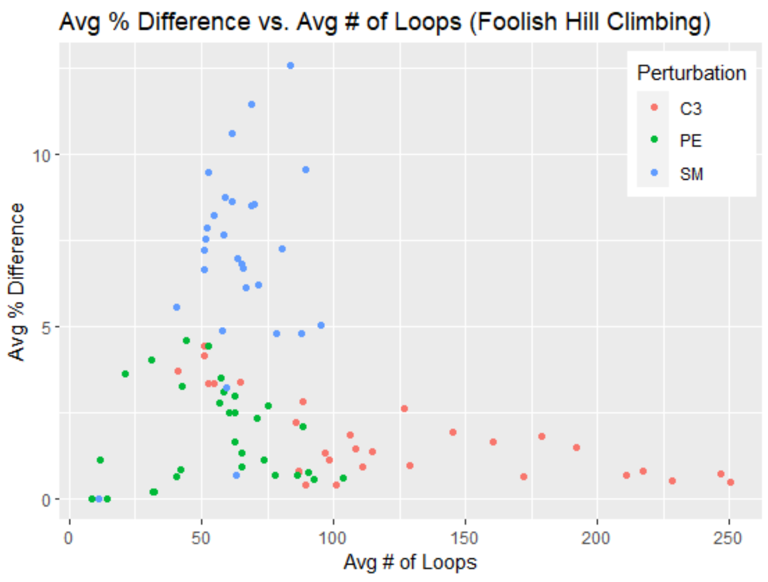


Figure 3: Average percent difference vs. average number of loops for foolish hill climbing.

Perturbation	% Difference	% Optimal	# of Loops	Time (s)
C3	**1.74**	2.67	125.	41.0
PE	1.86	**22.7**	**56.1**	**8.24**
SM	6.75	7.33	61.8	8.26

Table 4: Average performance of each perturbation in foolish hill climbing.

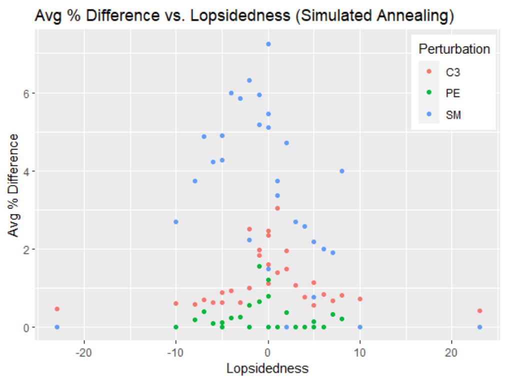
Table 4 shows how each perturbation function fared over all the tests. Clearly, PE is the top performer, as it is the best in three categories and a close second in the fourth. While C3 is more accurate than SM on average, the latter finds the optimal answer more often than the former. This is due to the fact that SM excels in extremely lopsided scenarios, while C3 does not take advantage of any particular grid shape.

4.2 Simulated Annealing

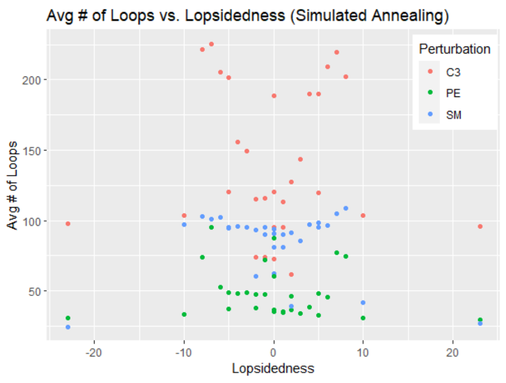
Figure 4 shows the effect of lopsidedness on performance for simulated annealing. There does not appear to be as much linear asymmetry in the data as there is in hill climbing. Another observation to note is that the data in Figure 4a appears to take on the shape of a bell curve about zero. This seems counter-intuitive, since square grids have more optimal possibilities. One theory for this could be that hill climbing and simulated annealing only utilize a single solution state, and these optimal configurations are not "near" each other in the search space. It may help to imagine the search space as a cube, where the eight vertices represent the eight optimal permutations for square grids. These vertices should theoretically pull the solution toward them due to their optimality and the optimality of the search space around them. Since only a single point within that search space is stored, it can get pulled between several of the vertices and get stuck in the middle of the cube or squarely in the center of one of its faces, far from an optimal solution.

Figure 5 shows the effect of the number of genes on performance in simulated annealing. Here, the biggest difference between simulated annealing and hill climbing is that PE is now clearly more accurate than C3 and quicker than SM, on average.

Figure 6 shows the performance of each perturbation function for simulated annealing. This graph draws the same conclusions as its hill-climbing complement, though PE is now more tightly clustered in the bottom left,

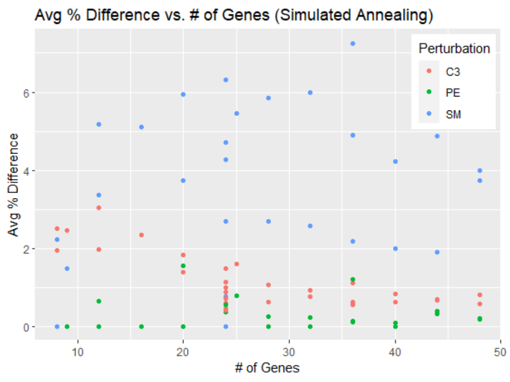


(a) Average percent difference.

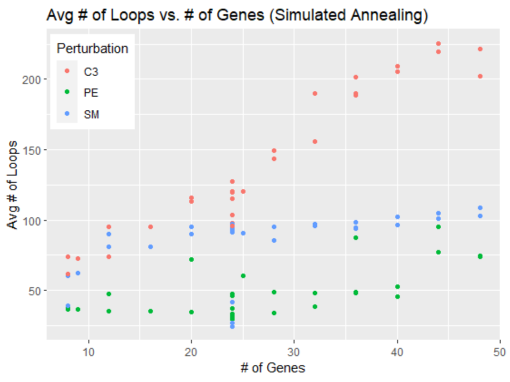


(b) Average number of loops.

Figure 4: Lopsidedness analysis for simulated annealing.



(a) Average percent difference.



(b) Average number of loops.

Figure 5: Problem size analysis for simulated annealing.

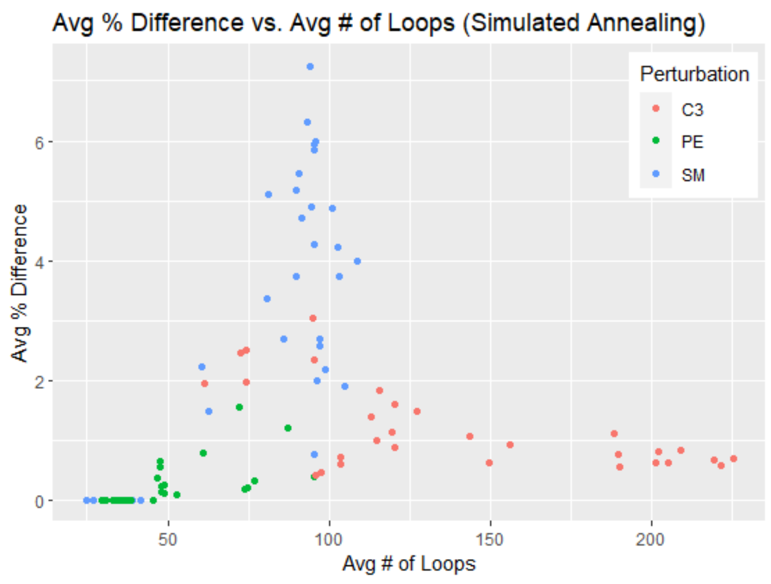


Figure 6: Average percent difference vs. average number of loops for simulated annealing.

Perturbation	% Difference	% Optimal	# of Loops	Time (s)
C3	1.19	10.0	140.	78.5
PE	**.235**	**81.3**	**48.7**	**12.8**
SM	3.45	18.7	84.5	22.9

Table 5: Average performance of each perturbation in simulated annealing.

signifying better relative performance.

Table 5 shows the average performance for each perturbation function across all tests. PE is now the top performer in every category. In fact, it found an optimal solution at least once in **every** testing scenario. C3 is again more accurate but less often optimal than SM. Overall, simulated annealing provides better accuracy and is more often optimal than hill climbing.

4.3 Genetic Algorithm

Figure 7 shows the effect of lopsidedness on performance in the genetic algorithm. The first interesting observation is that the shape of the data has flipped from a bell curve to a quadratic curve that is concave up. This

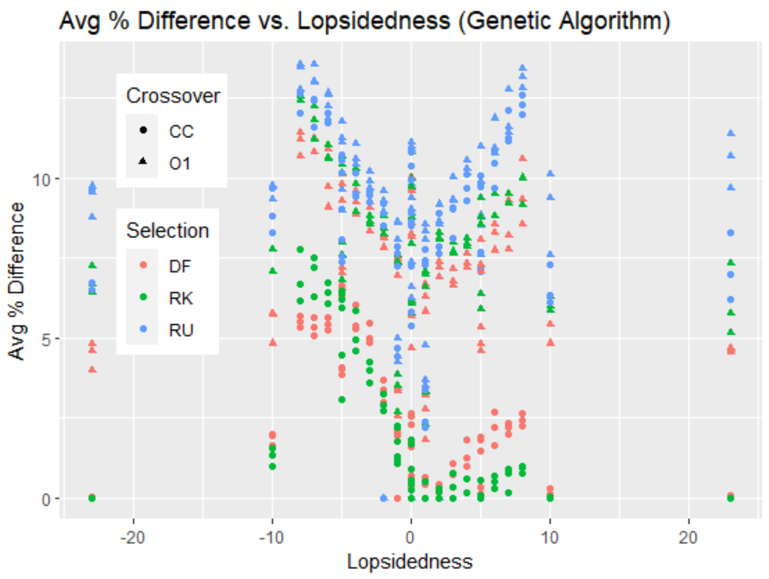


Figure 7: Average percent difference vs. lopsidedness for the genetic algorithm.

agrees with the hypothesis that it is easier to find solutions for square grids since they have eight optimal configurations; the more lopsided a grid is, the fewer the optimal solutions, and thus the harder it is to find them. Using the same theory discussed earlier, this could be due to the fact that the genetic algorithm stores many individuals at once, so there can be multiple sub-populations all nearing different optimal solutions. This would ensure at least some individuals are always near a global optimum and not being pulled away by a different global optimum that is much further away in the search space. The second observation is that the DF and RK selection methods, when paired with CC, perform significantly better with tall and thin grids.

Figure 8 shows how the performance of the genetic algorithm changes as a function of execution time. In general, the genetic algorithm takes longer with a larger problem size, so it indirectly shows the change in percent difference as the number of packages increases. Between this figure and the previous one, it is clear to see that any configuration which uses O1 for crossover or RU for selection underperforms greatly. In contrast, any configuration with CC and either DF or RK perform significantly better.

Table 6 shows the results of each genetic algorithm configuration averaged

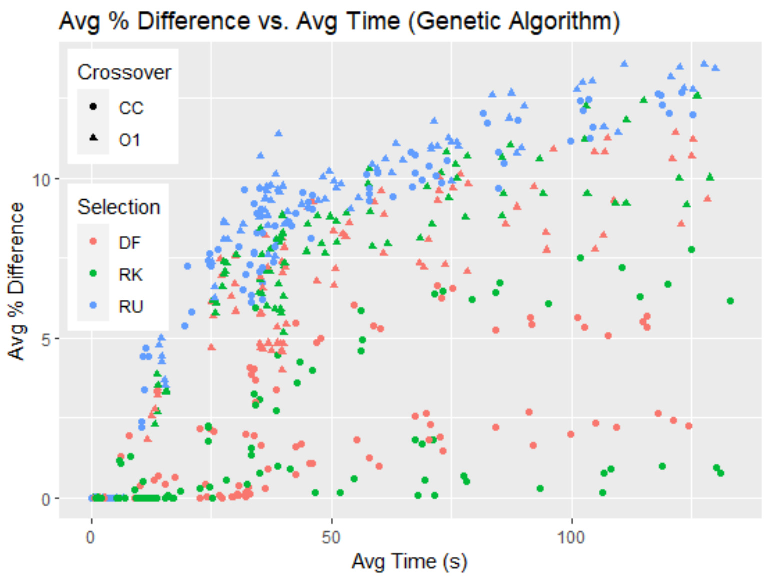


Figure 8: Average percent difference vs. average time for the genetic algorithm.

across all 30 different grid shapes. The combination of RK, CC, and PE achieves the highest optimality percentage and lowest percent difference. It also does so in the fewest number of generations and nearly the quickest amount of time. In general, the mutation functions have very little difference between them. However, the only two suitable combinations of selection and crossover strategies are DF-CC and RK-CC. On top of attaining little accuracy, O1 also consistently took longer to complete than CC.

However, DF and RK, when paired with CC, perform much better in positively lopsided scenarios, so Table 7 shows those results. Since a solution to any positively lopsided grid also yields a solution to the symmetric, negatively lopsided scenario, these results are still valid for comparison. RK, CC, and PE are now the best combination by far, averaging a small fraction of a percentage difference and almost always yielding an optimal answer. In fact, this is the lowest percentage difference that any algorithm has achieved in these tests, and it is optimal almost as often as simulated annealing with PE. However, the former takes nearly three times as much time to finish as the latter. Plus, the genetic algorithm fails to find even one optimal solution among its five tries in some of the larger scenarios.

Configuration	% Difference	% Optimal	# of Gens	Time (s)
DF CC C3	2.18	28.0	81.6	46.3
DF CC PE	1.99	36.0	79.1	46.8
DF CC SM	2.05	34.7	78.7	46.4
DF O1 C3	6.57	12.0	90.3	52.6
DF O1 PE	6.61	10.7	90.2	51.7
DF O1 SM	6.55	10.7	90.7	51.8
RK CC C3	1.77	43.3	70.2	**42.7**
RK CC PE	**1.61**	**54.0**	**66.1**	44.4
RK CC SM	1.97	42.7	69.0	**42.7**
RK O1 C3	7.23	10.7	90.5	51.8
RK O1 PE	7.41	10.0	90.5	52.1
RK O1 SM	7.22	10.0	90.5	51.0
RU CC C3	7.96	10.7	91.4	48.1
RU CC PE	7.94	10.7	90.8	48.4
RU CC SM	7.99	10.7	91.2	48.8
RU O1 C3	8.76	10.0	91.3	51.8
RU O1 PE	8.75	10.0	91.7	51.9
RU O1 SM	8.88	10.0	91.5	51.9

Table 6: Average performance of each configuration in the genetic algorithm.

Configuration	% Difference	% Optimal	# of Gens	Time (s)
DF CC C3	1.05	35.3	76.9	44.0
DF CC PE	.937	45.9	74.7	44.6
DF CC SM	.895	47.1	71.9	43.6
DF O1 C3	6.06	14.1	88.5	51.0
DF O1 PE	6.11	11.8	89.0	50.8
DF O1 SM	5.84	12.9	89.3	50.1
RK CC C3	.413	61.2	58.1	**37.8**
RK CC PE	**.208**	**80.0**	**52.8**	38.2
RK CC SM	.396	60.0	58.0	38.4
RK O1 C3	6.67	12.9	89.0	50.9
RK O1 PE	6.85	11.8	88.8	50.5
RK O1 SM	6.60	11.8	89.0	49.8
RU CC C3	7.37	12.9	90.5	46.5
RU CC PE	7.55	12.9	89.1	46.3
RU CC SM	7.50	12.9	89.9	46.7
RU O1 C3	8.35	11.8	90.0	50.3
RU O1 PE	8.43	11.8	90.9	50.2
RU O1 SM	8.45	11.8	90.4	51.1

Table 7: Average performance of each configuration in the genetic algorithm for grids with non-negative lopsidedness.

5 Conclusion

This paper tested “foolish” hill climbing, simulated annealing, and a genetic algorithm against the two-dimensional package placement problem. It found that simulated annealing performed the best overall, especially when paired with the pairwise exchange perturbation function. This combination achieved optimality at least once in every test scenario. When using rank for selection, cut and crossfil for crossover, and pairwise exchange for mutation, the genetic algorithm achieved a slightly lower percentage difference than simulated annealing in certain lopsided scenarios. However, the genetic algorithm found an optimal solution less often and took more time to complete.