

Genetic Algorithms Applied to the Freeze Tag Problem

Blaine Hoffman
Computer Science Department
Denison University
Granville, OH 43023

Abstract

This paper describes the effectiveness of Genetic Algorithms in application to the Freeze-Tag Problem with focus on its relative performance compared to existing and previously tested algorithms. Genetic Algorithms lead to improvements over existing solutions via simulating genetic evolution, the overall goal to return generally better solutions.

I. Introduction

The Freeze-Tag Problem (FTP) deals with scheduling and parallel processing. The problem seeks to find the fastest ordering to awaken a set of inactive nodes. Solutions are represented by awakening schedules, representations of this ordering, that describe how the work is completed in awakening all principals in the problem, be they robots or nodes on a network. We will discuss the problem in more detail in the next section.

Previous research has tackled this problem primarily using greedy choice strategies. While these results are fairly good, there is certainly room to look into the varying factors of a solution and its creation in order to attain better results. This research project set out to do this using Genetic Algorithms. As is described in Whitley's Genetic Algorithm tutorial [7] and Aarts and Lenstra's *Local Search in*

Combinatorial Optimization [1], these algorithms offer improvements on existing solutions. To accomplish this, the algorithm treats solutions as chromosomes in a gene pool, testing their fitness and genetic viability compared to one another as well as providing means for genetic variation through recombination and mutation. This yields improved solutions from the initial generation, and the algorithm processes many generations before returning a result.

Using a Genetic Algorithm, we evaluated awakening schedules for the FTP and compared them to solutions given by greedy algorithms and a center-of-mass algorithm. The next section describes the FTP in detail. The third section runs through the Genetic Algorithm as it applies to solving this problem, followed by our results and data. We end with some remarks concerning potential improvements to Genetic Algorithms applied to the FTP.

II. The Freeze-Tag Problem

The Freeze Tag Problem comes from the field of swarm robotics and has been described in detail by Arkin et al [2][3] and Sztainberg [6]. In the FTP, there is a set of n robots in which each robot is a node or point on the Cartesian plane. An example is shown in **Figure 1**. Initially, only one of these n robots is awake, or active; the other $n - 1$ robots are still dormant. The goal is to awaken the dormant robots so that all n

This research project was made possible by a grant from the Anderson Endowment Program. Suggestions, background information, advice, and other guidance were provided by Dr. Matt Kretchmar, Dr. Kevin Hutson, and Dr. Todd Feil throughout this project.

robots are awake in the quickest time possible. In order for an active robot to awaken another, it must physically travel to the other's location and make contact. Once awakened, this other robot can then aid in awakening the remaining dormant robots. Thus, the process cascades until all of the robots are active. For calculations, one unit distance is assumed to cost one unit time to travel.

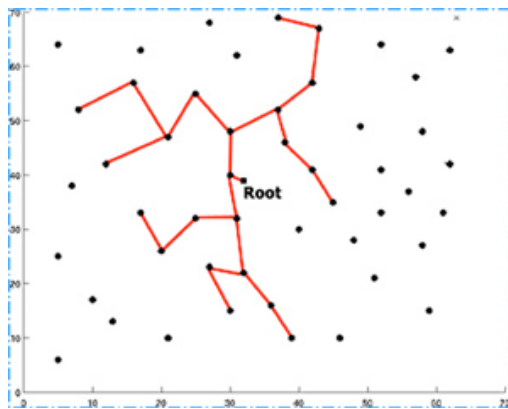


Figure 1 – Awakening Schedule Creation – For all robots on a given plane, more are added to the schedule until all are connected.

The FTP is similar to the well-known Traveling Salesman Problem (TSP). In the TSP, our salesman must choose a path that visits every city (consider the n robots) in the region (consider the Cartesian plane) within which he works, returning to his city of origin. The optimal solution makes this circuit in the shortest time and, as with the FTP, the shortest distance. The FTP is a modification of this problem, adding a level of complexity [3]. Consider that the salesman is allowed to hire another salesman within each city he visits; this reflects the interaction of the robots [3]. Once the salesman moves on to another city, his hired help will also travel to a different city; both hire help in the new cities. Eventually, there will be several salesmen traveling in parallel with one another, creating multiple paths.

Likewise, the robots travel in these parallel paths as they awaken those that remain asleep. However, there is no restraint forcing them to return to their cities of origin. These parallel paths, add to the complexity when searching for optimal or good solutions.

The FTP gets its name from the game of Freeze-Tag. In this game, one player is designated as "it," the others becoming frozen when touched by this player. In order to become unfrozen, a "free" player must touch them; once unfrozen, a player may then also free frozen players. The FTP represents the case in which all but one player are frozen, this lone player free to unfreeze them ($n - 1$ dormant robots, with one active to awaken them all). The optimal solution is to find the best awakening schedule, representing the best ordering to unfreeze the players, enlist their aid to free others, and complete these tasks in the shortest time [6].

Awakening schedules, a listing of the order in which each robot is activated and by which robot this occurs, represent solutions to the FTP. The structure of this data type resembles a binary tree, as each robot's location can lead to two others (this is because, to awaken a robot, another must have come into contact with it, placing two at that location) and is illustrated in **Figure 2**. The exception is the root, or initially active robot, for there is only one robot at this location. This schedule also records the robot that awakened any particular robot (NULL for the root robot), the one or two robots scheduled to be awakened by either of the two robots now present at that point, and the current makespan of the schedule at that point. Makespan here is the calculation of the longest distance (and, thus, time) from a node to any of the leaves that is its descendent.

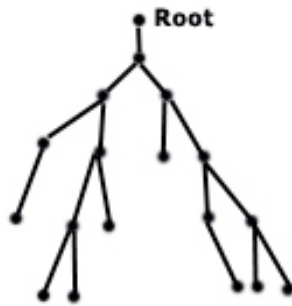


Figure 2 – Sample Awakening Schedule Structure
 – The awakening schedule is stored in what resembles a binary tree structure

This problem has applications in broadcasting and network design, as both relate to efficient transportation of information [2]. It contains elements of routing, as well, in regards to the scheduling of the multiple robots' travel and paths to varying locations while passing information. This also implies the importance of scheduling in this problem, as this task gains difficulty as the number of awakened robots working in parallel increases [3]. Much like with parallel processors, an efficient schedule must be found.

The FTP has been shown to be NP-Hard by Arkin et al [2] and affirmed as such by others [3][6]. Therefore, research has looked into applying various approximation algorithms to the FTP. These algorithms do not typically find the optimal solution, instead returning valid solutions that are considered good – not optimal, yet relatively close and practical. The advantage of using approximation algorithms, of course, is that they *can* run in reasonable, polynomial time.

III. Prior Research

The initial research of the FTP has yielded proofs that the problem itself is NP-Hard. In [2], Arkin et al make the reduction from N3DM, while they use 3SAT in [3]. The research then provides several heuristics for

solving the FTP, typically laying emphasis on greedy strategies. In addition, the various potential shapes of the region containing the robots affects the abilities of an algorithm to find good solutions [2].

Perhaps shown to be most successful is the greedy algorithm. As is typical of algorithms that employ a greedy choice, this algorithm works via choosing closest robots to add to the schedule. This means that, for any active robot, it chooses the closest dormant robot to awaken next [6]. In some cases, the robot may head toward a robot that will be awakened by a different active robot before it completes its journey; this leads to inefficient scheduling. To account for this, the greedy algorithm can be altered to dynamically change paths of waking robots [6]. Thus, there are two greedy algorithms applicable to the FTP – the Fixed (composes one schedule and runs it) and the Dynamic (alters a robot's path on-line if another robot has already awakened the robot at its destination).

Intuitively, this greedy choosing is a logical choice in order to awaken robots quickly, as the algorithms select edges in non-decreasing order. However, there are cases within which this method of selection does not yield good solutions. For example, Arkin et al show that star graphs, in which all edges connect to the same central point, can at best reach a ratio of 7/3 in relation to an optimal solution [2]. For certain weighted graphs, in which the time requirement for traversing an edge is its weight, this ratio is improved to 5/3. This occurs because of the nature of greedy choices placing the longest edges last in schedules; instead of sending a robot to these edges in tandem with closer paths, the longer paths (and thus times) are added to the schedule's makespan by being placed later in the schedule. Choosing differently may improve the result, if not yield an optimal solution. The inability to address

collections of separated clusters of robots, instances in which making a non-greedy choice may be optimal, in open graphs on the Cartesian plane suggests improvements can be made. It should also be noted that greedy algorithms will always yield the same awakening schedule for a given root on a graph.

IV. Genetic Algorithms

Genetic algorithms require that the initial generation be comprised of viable solutions to the problem toward which they are applied. Thus, with this initial generation of solutions (chromosomes), the evolutionary algorithms can simulate genetics as they work towards improvements of the N given solutions, the goal being to find a very good solution due to evolution over generations. To this end, our algorithm needed a set of solutions to the FTP at the start. The initial generation was seeded with N awakening schedules given by such algorithms as the greedy-fixed, greedy-dynamic, or even random, unbalanced awakening schedules. Before our algorithm initiated its genetic process, these solutions were placed into the generation structure, a data type that contained an array of chromosomes. The chromosomes were arrays of n (n to reflect the n robots; N represents the number of solutions in a generation) nodes representing the awakening schedule. Each node represented a robot and contained the ID of the robot that awakened it, the makespan of the awakening schedule from its position down, and the ID of the one or two robots awakened from this position (both the now-awakened robot and its contact proceed from this position to awaken others).

After the initial generation was created, the algorithm simulated the following steps in order to reach the best solution it can reach via improvements on

the current generation: evaluation, fitness and selection, reproduction and cross-over, and mutation. The following is a basic outline of the steps of the algorithm:

GeneticFTP

```

for  $i \leftarrow 0$  to  $N$  chromosomes in a generation do
(Evaluation)
  Evaluate[ $i$ ]  $\leftarrow$  makespan[ $i$ ]
  AverageEvals  $\leftarrow$  AverageEvals + Evaluation[ $i$ ] / 2
for  $i \leftarrow 0$  to  $N$  chromosomes in a generation do
(Fitness and Selection)
  fitness[ $i$ ]  $\leftarrow$  AverageEvals / Evaluate[ $i$ ]
  while fitness[ $i$ ]  $\geq$  1 do
    passed[ $i$ ]++
    fitness[ $i$ ]--
  while numberPassed <  $n$  do
    tournament-style selection (see [7])
for  $i \leftarrow 0$  to  $N$  chromosomes in a generation do
(Recombination and Cross-Over)
  if 0.7 probability is met
    create child from two parents
    replace recipient parent with child
for  $i \leftarrow 0$  to  $N$  chromosomes in a generation do
(Mutation)
  if 0.1 probability is met
    randomly swap two edges, provided a viable
    schedule is created
  
```

This process repeats for P generations.

Evaluation

The evaluation stage value was straightforward, based upon the makespan value of an awakening schedule. We then calculated the average of all of these makespans. This and the array of evaluations were then passed right into the fitness and selection stage.

Chromosome	Evaluation	Fitness
1	25.097	0.8001
2	22.0347	0.9113
3	17.2978	1.1608
4	14.875	1.3499
5	21.096	0.9518

Table 1a – Evaluation and Fitness Example

Fitness and Selection

Given the evaluation of the solutions and their average, this stage determined how many copies of each solution passed onto the subsequent generations. First, the fitness values of all chromosomes were calculated. An example of this can be seen in **Table 1a**. Because we sought a minimal makespan (meaning a much faster awakening schedule), a chromosome’s fitness value was the average evaluation of all solutions within a generation divided by its evaluation. Thus, $fitness[i] = AverageEvals / Evaluate[i]$, where i was the current solution considered. Those with fitness greater than or equal to one passed one copy for each time one could be subtracted from their fitness number. Their remainders and the fitness values of the other chromosomes were then chosen tournament style [7], probabilistically selecting those deemed better. **Table 1b** uses the numbers from the fitness example above in application to this stage. Some solutions may pass several copies, others none at all. This will eliminate those chromosomes whose awakening schedules were inferior and deemed less fit.

Selection (Initial)	Fitness Remainder	Selection (Final)
0	0.8001	0
0	0.9113	1
1	0.1608	1
1	0.3499	2
0	0.9518	1

Table 1b – Example of Selection Process

Reproduction and Cross-over

This stage took care of copying chromosomes from the existing generation into the subsequent generation. Of course, the selection process from the prior stage determined which chromosomes are passed along; however, this stage also applied genetic recombination, creating new

awakening schedules from the available genetic data provided. As a result, the evolution of genes is presented into the set of all awakening schedules, providing improvements to the solutions in further generations. This process is the most vital in this genetic algorithm. Typically, it is recommended to keep the rate of reproduction and cross-over around seventy-percent, as work with Genetic Algorithms shows this to be somewhat optimal [7] [1]. This means that there was a seventy-percent probability for each pair of parents to “mate” and create a child solution.

In this stage, one parent was the recipient, the other the donor; the recipient was the parent whose makespan was larger. The two parents’ set of genes, awakening schedules in our case, combined to create the child awakening schedule (see **Figure 3**). This method was adapted from evolutionary algorithms used in Minimum Spanning Tree problems, as described in [4] and [5]. During this process, the child created from the parents’ genes must go through three stages. First, the child must contain all edges present in both parents. Once these were added to the awakening schedule of the child, it then connected each remaining robot individually, taking the smaller of the two options the parents provided. In some cases, the parents may have different robots set as their roots; if this occurred, the child’s root was selected from the donor parent, as its overall makespan was less. Lastly, if any robots were not connected to the awakening schedule in the prior steps, typically due to incompatibilities of the parents’ schedules that would result in loops or invalid schedules, the algorithm searched top-down for the first available slot to schedule the robot’s awakening. This means it attempted to place these robots into the schedule earlier as opposed to later, preventing unnecessary additions to the overall makespan.

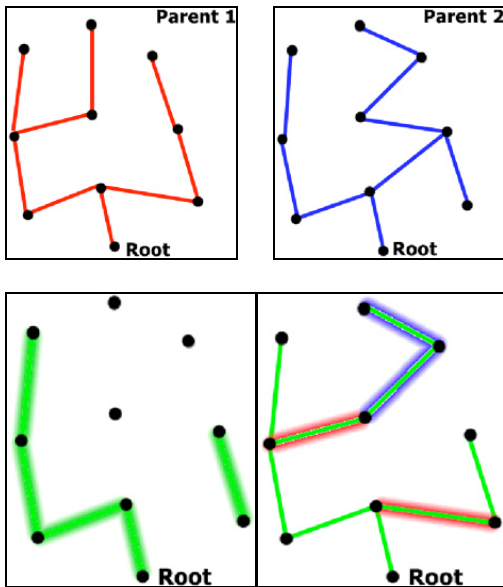


Figure 3 – Formation of a Child Awakening Schedule From Two Parents. The child first takes all edges common to both parents. It then takes the edge from the parent that best connects each robot individually to the overall awakening schedule.

Once these additions completed, the child schedule was created. It was copied overtop of the recipient parent, overwriting the more costly makespan with this new and generally better makespan. In each case, the child created yielded a makespan as good as the better of the two parents or smaller. This process repeated for all potential pairs of parents throughout the intermediate generation. These children created were copied into the intermediate generation. At this point, this original generation structure had not been updated to reflect the changes in neither the selection stage nor those in recombination. In order to update these changes, we copied the intermediate generation overtop of the generation structure.

Mutation

As with any genetic process, there is always

a chance for mutation, in which genetic structures are altered not by parental combination and cross-over but by random alterations to genes. Similar to the probability of recombination, mutation is typically set at one percent [7]. For each chromosome in a generation, if this condition was met, mutation occurred. For the FTP, the mutation would cause a swap of edges between nodes in an awakening schedule.

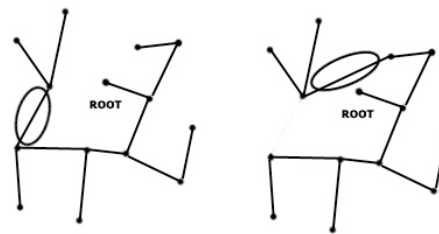


Figure 4 – Edge Swap in Mutation – The algorithm arbitrarily removes and edge in favor of placing a new edge into the schedule.

There were two methods to accomplish this task, one being to make a random swap and the other being a search through neighborhoods. Each must account for loops and insure that the mutation still provides a valid schedule. The latter method, however, probabilistically searched for edge swaps that would result in a somewhat longer makespan. Each provided potentially worse awakening schedules after a mutation. The goal of mutation is to do exactly this, as potentially worse solutions help an improving algorithm climb out of local minima while searching for its best solution. The former method took longer to run due to needed error checking and was thus not used in the final tests.

Once all stages had been run, the global minimum schedule was maintained, as well as its makespan, and the ongoing average makespan over all generations was updated. Additionally, the minimum

makespan of the current generation was calculated. These values allowed us to compare the current generation with prior values, and they highlighted the trend for the average makespan to decrease gradually over generations while the best periodically decreased more dramatically. Moreover, because of the probabilistic nature of the algorithm and the factors of mutation, it was possible for a subsequent generation to contain less-desirable solutions to the FTP. While this is useful for climbing out of local minima, we did not wish to discard an answer that was globally better. Once a generation had gone through evaluation, fitness and selection, recombination and cross-over, and mutation, it began the process over again, starting at evaluation. This method repeated as the algorithm approached its best solution.

V. Data and Analysis

It was our goal to improve upon existing results with the Genetic Algorithm, thus finding valid awakening schedules with shorter makespans than those found with greedy algorithms, for example. We used the greedy fixed, greedy dynamic, and center of mass strategies as a basis of comparison for the performance of the genetic algorithm. The algorithm was tested on the following data sets, readily available in the TSP Library: eil51, eil76, kroA100, d198, lin318, and att512 [8]. In addition, we used data sets we created, labeled twoCirc100, tenCirc300, and moon1000. These were chosen because of their shapes and clustering of nodes, thus giving us more varied conditions for awakening robots. As mentioned earlier, the greedy choice may not be the best choice when dealing with multiple clusters, and we hoped to improve upon such choices with the evolutionary simulation.

For each data set, the genetic algorithm program was run twenty times in order to calculate the average of the algorithm's performance and check for consistency. For purposes of comparison, the best schedule found was selected for the data table. The algorithm was run on each of the listed TSP data sets with a generation size N of 200, meaning each generation had 200 chromosomes (solutions) within itself. Subsequently, we ran the algorithm with a generation size P of $2n$, or twice the number of robots within the data set; for kroA100 and twoCirc100, we did not have to run this alteration, as $2n$ in these cases is 200. Increasing the population size of a generation did show an improvement to the algorithm's solutions. The data can be seen in **Table 2** at the end of this paper.

To seed the initial generation's chromosomes, we used a function that created random awakening schedules with no regard for optimality or balancing. In other runs, we seeded this generation with schedules from the greedy dynamic function; it is important to note that this function will always return the same awakening schedule for a specific root, or initial, robot. To account for this, random roots were sent to the greedy algorithm in creating these seed schedules. It was found that this, on average, did not improve the makespans of the awakening schedules found by the genetic algorithm. This indicated that the genetic algorithm's ability to improve on solutions via evolution and genetic variation was more accurate than the greedy algorithms. What this means is that the genetic algorithm could improve given seed generations to an extent that surpassed the ability of greedy choice strategies to answer the FTP regardless of what methods were used to initiate the evolutionary process. Additionally, the percent chance of recombination and that of mutation, noted as typically being set to 0.7 and 0.01

respectively, were altered to 0.8 and 0.05 in other runs. The data reflected no significant change in the ability of the algorithm to find faster awakening schedules.

The ability of an algorithm to solve the FTP can be measured by how well the makespan of its best awakening schedule compares with the radius of the problem space. Because the robots must travel to other robots to awaken them, the best awakening schedule cannot be better than the radius of the total space of the problem, keeping in mind that one unit distance translates into one unit time. Another comparison of these results was made, this time dividing the best awakening schedule's makespan by the radius of the data set. In this manner, the solutions have been normalized for comparison. These results can be seen in the **Graph 1** at the end of this paper. The better of the two cases of the genetic algorithm is represented.

For all data sets tested, the genetic algorithm proved to yield smaller makespan values than previously-tested algorithms. While these improvements may appear to be slight, especially given the improvement on greedy fixed by greedy dynamic, they are not insignificant. Similar to work with the TSP [1], there exists a lower bound to these problem instances; as algorithms find solutions closer and closer in value to these optimal solutions, the rate of improvement is more difficult. This is typically true of any approximation algorithm, as providing the means to find better solutions clashes with avoiding complexity and excessive running times. With this in mind, we felt the improvements upon the well-performed greedy Dynamic algorithm are worthwhile and indicate the possibility of genetic algorithms to be applied more successfully to the FTP in the future. Also, as mentioned previously, performing better than the greedy algorithms, even slightly, suggests that evolutionary algorithms are a better

choice for solving the FTP.

The algorithm was encoded in C++, compiled and run on both 1 GHz Pentium-III machines running Red Hat Linux and Apple 1.2 GHz Power Mac G4s running OS X. Running time ranged from twenty to thirty seconds on the smaller data sets (eil51, eil76) to about thirty or forty minutes on larger data sets (att532, moon1000) for one run.

VI. Conclusion

The genetic algorithm was shown to improve upon the awakening schedules of prior applications of the FTP. Due to its evolutionary processes and genetic recombination, the algorithm is able to make non-greedy decisions and arrive at a faster solution. Those schedules that are slower and, thus, undesirable are deemed unfit and fail to survive on to future generations. Also unlike greedy choice methods, the mutation phase allows for seemingly negative connections in the awakening schedule to be made that may potentially result in quicker overall solutions to the problem.

The most essential portions of the genetic algorithm are the recombination and cross-over stage and the mutation stage. These evolutionary simulators are the driving force of increasing the fitness of a generation (the former) and aiding in the search throughout all neighborhoods instead of just one (the latter). Further improving upon these algorithms could result in even better solutions to the FTP. Perhaps better conditions for choosing how to construct a child schedule from two parents could be offered, or a smarter search through potential swaps within the mutation function could yield a more intelligent genetic alteration. Alterations to the population size of a generation or the conditions with which evolutionary simulation ends may also

provide further improvements to solutions.

Other optimizations should also increase the performance of the genetic algorithm. As discussed in Aarts and Lenstra [1], there are separate optimization routines that can be merged into the structure of a genetic algorithm to improve the results returned. Such implementations, as their work on the TSP problem indicates, would narrow the gap between potential lower bounds to the FTP and the solutions given by a genetic algorithm.

VII. References

- [1] Aarts, Emile and Jan Kaerel Lenstra. *Local Search in Combinatorial Optimization*. John Wiley and Sons, Ltd. 1997.
- [2] Arkin, E. M., Michael A. Bender, Sandor P. Fekete, Joseph S. B. Mitchell, and Martin Skutella. "The Freeze-Tag Problem: How to Wake Up a Swarm of Robots." In *Proc. 13th ACM-SIAM Sympos. Discrete Algorithms*. pp. 568 – 577, 2002.
- [3] Arkin, E. M., Michael A. Bender, Dongdong Ge, Simai He, and Joseph S. B. Mitchell. "Improved Approximation Algorithms for the Freeze-Tag Problem." *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*. pp. 295 – 303, 2003.
- [4] Julstrom, Bryant A. and Gunther R. Raidl. "A Permutation-Coded Evolutionary Algorithm for the Bounded-Diameter Minimum Spanning Tree Problem." *2003 Genetic and Evolutionary Computation Conference's Workshops Proceedings, Workshop on Analysis and Design of Representations*. pp 2-7, 2003.
- [5] Raidl, Gunther R. "An Efficient Evolutionary Algorithm for the Degree-Constrained Minimum Spanning Tree Problem." *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*. 2000.
- [6] Sztainberg, Marcelo O., Esther M. Arkin, Michael A. Bender, and Joseph S. B. Mitchell. "Analysis of Heuristics for the Freeze-Tag Problem." *Proc. Scandinavian Workshop on Algorithms*, Vol. 2368 of *Springer-Verlag LNCS*, pp. 270 – 279, 2002.
- [7] Whitley, Darrel. "A Genetic Algorithm Tutorial." *Statistics and Computing* (4). pp. 65 – 85, 1994.
- [8] TSPLIB. <<http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>>

Table 2 – Comparison of Genetic Algorithm to Prior Strategies

	Greedy Fixed	Greedy Dynamic	Center of Mass	Genetic Algorithm w/ P = 200	Genetic Algorithm w/ P = 2n
eil51	75.03	61.63	55.92	57.2635	54.7879
eil76	68.7	56.64	59.65	51.7207	54.3425
kroA100	3857	2515	2591	2445.27	*
d198	3087	2433	2446	2392.84	2396.86
lin318	4612	2806	2944	2765.1	2759.29
att532	9720	5096	5367	5069.19	5064.18
twoCirc100	16.485532	11.819027	11.265477	11.041	*
tenCirc300	446.686769	238.569081	231.789679	229.179	225.674
moon1000	3.059198	1.823404	1.890797	1.82	1.81939

* not run, as 2n = 200

Graph 1 – Comparison of Algorithms Using Makespan/Radius

