A Multi-objective Genetic Algorithm for Employee Scheduling

Russell Greenspan
University of Illinois
December, 2005
rgreensp@uiuc.edu

ABSTRACT
A Genetic Algorithm (GA) is applied to an employee scheduling optimization problem with varied, competing objectives and thousands of employees. An indirect chromosome encoding is used with genetic operators based on general GAs [13] and Evolutionary Strategies [6]. Population fitness is determined via a weighted-objective function, featuring a Balance Factor that encourages or disregards equality among the objectives. Initial results are promising.

1. INTRODUCTION
The company I work for is the leading provider of online tutoring services, employing over 1,000 tutors and providing tutoring services to over 4,000 students each day. Each of the company's 1,000+ tutors is trained in one or more of the four topic categories offered (Math, Science, English, and Social Studies). To maximize the company's efficiency and minimize cost, the company must optimally schedule these 1,000 tutors each week, while ensuring that competing factors – the company's needs and tutors' needs – are fairly managed.

This paper explores using a Genetic Algorithm (GA) to schedule the company's tutors. The algorithm generates weekly schedules with hourly granularity. It seeks to maximize five competing factors, namely total and average coverage of the four topic categories, equal distribution of hours amongst employees, scheduling of employees in chunks of time, and scheduling of employees to time they have requested. While the test data was drawn from real-world data, the algorithm is general enough in design to accommodate similar employee staffing scenarios.

2. RELATED RESEARCH
Since scheduling problems are generally NP-hard and were often optimized by heuristics alone, they have made a good target for GA research and much progress has been made in using GAs for scheduling optimization. However, I believe the size of the search space and the mixture of competing factors at the company adds a number of unexplored elements to the problem space.

A brief synopsis of relevant research includes [1], which took a hybrid approach to a similar problem on a smaller scale, using a GA to create monthly employee schedules for 25 or fewer employees. However, the hybrid component – "repair operators" that iteratively adjust the population – decrease its value as a GA. [2] used a permutation-based encoding approach similar to my solution, with a shift-based 'schedule-builder' that iteratively assigns employees to uncovered shifts. The algorithm accounted for different nurse grades (similar to my concept of coverage weight described herein), but again though, the algorithm was developed for a maximum of 30 employees (nurses). [4] used a similar weight-based fitness calculation for a relatively simple scheduling problem.

Regarding general scheduling optimization problems, [7] and [8] provide good overviews of existing techniques, and thorough statistical approaches to queue management are outlined in [11] and [12].

This paper assumes a solid understanding of the various factors involved in GAs [13].

The rest of this paper is organized as follows: encoding of chromosomes in my GA is discussed in section 3, input variables and implementation details of the GA are described in sections 4 and 5, and test data and results are described in sections 6 and 7.

3. CHROMOSOME ENCODING
Company administrators set different staffing levels for each day-hour combination in a weekly schedule of the company's tutors. If coded directly, each weekly schedule with hourly granularity of 1,000 employees would consist of 168,000 genes (7 days x 24 hours x 1,000 employees). In such an encoding, each gene would indicate whether or not employee *x* is scheduled for day-hour slot *y*. Simulation of evolution with such large chromosomes would be very inefficient and time consuming (just ask nature), and genetic operations like crossover would produce many invalid schedules.

Instead, we can indirectly represent each one week schedule by a 24 hour x 7 day grid and a permutation of scheduled employees in each slot. This grid is encoded in a chromosome with 168 positions; a position's index corresponds to one of the 168 day-hour slots in the following manner:

	SUN	MON	TUE		SAT
12am – 1am	0	24	48	…	144
1am – 2am	1	25	49		145
2am – 3am	2	26	50		146
⋮	⋮	⋮	⋮		⋮
9pm – 10pm	21	45	69		165
10pm – 11pm	22	46	70		166
11pm – 12am	23	47	71		167

Thus position 0 corresponds to Sunday, 12am to 1am, position 1 to Sunday, 1am to 2am, etc., through position 167 which corresponds to Saturday, 11pm to 12am. Within each day-hour slot, we store a set of the unique IDs of employees to schedule:

Day-Hour Slot	0	1		167
Employee IDs	{212, 102, 938}	{223}	…	{102, 432}

So in essence, our final encoding is a permutation of employee IDs. Since each day-hour slot will have a significantly different number of employees to be scheduled, a chromosome contains only the data it uses (i.e. a day-hour slot consisting of 0 employees uses an empty encoding). In such an implementation each chromosome can have variable length, making this is a particularly efficient mechanism by which simulate evolution.

For output, I implemented a simple iteration over the collection of employees and spit out the assigned schedule for each employee. In this output, each employee's schedule contains 168 1-bit entries, with a 1 or 0 indicating the employee is scheduled or not for the day-hour slot corresponding to the bit's position. This output is essentially the direct encoding described above, and so with 1,000 employees, the complete schedule is output in 168,000 bits.

4. INPUT VARIABLES

4.1 Administrative

An Administrator specifies the following values:

- Min and max values of employees that can be staffed during each day-hour slot; a random number between each min and max value is then chosen for each day-hour slot when the GA population is created. This functionality is intended to provide an administrator minimum and maximum values to control staffing expenses.
- Weighted coverage values for each coverage area. The current implementation allows specification of up to four coverage areas.

4.2 Genetic Algorithm

The GA has the following inputs:

- Weights for each of the fitness values (Total Coverage, Average Coverage, Employee Distribution, Requested Slots Staffed, and Continuous Blocks Staffed).
- Balance weight, to specify how strongly to favor solutions that balance the different fitness values.
- Number of generations to run.
- Selection pressure to apply for the first 0 to 75th percentile generation, 76th to 95th percentile generation, and 96th to 100th percentile generation.
- Mutation rates to apply up to the 75th percentile generation and from then on.

5. IMPLEMENTATION

C# code: http://www.russellgreenspan.com/software/GA/code.zip.

5.1 Genetic Operators

I experimented with almost every combination I could think of for population size, selection factor, crossover types, and mutation rates. The following is a synopsis of the best results from my experiments.

5.1.1 Selection

I used Tournament Selection with varying selectivity as the GA proceeds. Tournament Selection with varying selectivity picks the best of *x* individuals at random, allowing a broad range of population members to be examined early on with a "tightening up" effect as the GA narrows in a solution. A selectivity factor below 1 indicates that with (1 – factor) probability, the loser of the tournament should be selected. With a population size of 500, I used selectivity factors of 1 up to the 75th percentile generation, 2 from the 75th percentile generation to the 95th, and 5 from the 95th percentile generation to the end.

I tried selection in two ways: selecting the best two from the parents and two children, and replacing the worst population member with the children if the child was fitter. The idea behind the first method is that the generated children are potentially better, yet very similar versions of their parents. By replacing the parent of a more potent child, we keep diversity high in the population. The idea of the second method is that we want to remove individuals from the population that offer us little help of solving the problem.

In practice I found that keeping diversity high was most important to finding the best solution. One reason for this is that the chromosomes in this problem space are extremely long, and a population member with a lower fitness value can easily contain a piece of the optimal solution while having an overall low fitness value.

5.1.2 *Crossover*

Crossover works by swapping day-hour slots between the two chosen population members. In the best case scenario, this has the effect of taking the best day-hour slots of the two chosen members, and in the worst case scenario the opposite is true. I experimented with both single-point and dual-point crossover and found that single-point crossover produced more fit populations.

For example, if we select the following two chromosomes:

Day-Hour Slot	0	1		167
Employee IDs	{212, 102, 938}	{223}	…	{102, 432}

Day-Hour Slot	0	1		167
Employee IDs	{818}	{201,101,102 }	…	{899, 902, 323}

and pick position 2 as the crossing point, we would have the following two children:

Day-Hour Slot	0	1		167
Employee IDs	{818}	{201,101,102 }	…	{102, 432}

Day-Hour Slot	0	1		167
Employee IDs	{212, 102, 938}	{223}	…	{899, 902, 323}

I also implemented three- and four-way crossover, where three or four parents combine to create two children in a method similar to Evolutionary Strategies [6], but this did not provide better fitness values so I stuck with the two-parent model.

5.1.3 *Mutation*

While crossover functions by moving whole day-hour slots without regard to the employees scheduled within, I implemented Mutation to alter the employees within the slots. I experimented with three different mutation techniques:

- Change the number of scheduled employees in the slot and remove or add employees to meet the new number.
- Change the number of scheduled employees and pick a new, random set of employees to schedule.
- Swap a random number of employees in each day-hour slot for other random employees.

In the end, however, the first technique produced more fit populations and was therefore chosen to be the mutation operator. I believe this is because the technique tends to leave positive characteristics in place and improve negative characteristics.

I also tried three variations of Mutation rate, the first with Mutation throughout as in a standard GA, the second as in Evolutionary Strategies [6] where the mutative factor decreases as the algorithm narrows in on a solution, and the third applying the mutative operator somewhere at the 75th percentile generation with a relatively high (0.01 to .1%) likelihood of mutation. The third approach was based on the assumptions that Mutation was best applied toward the end of the GA, thereby using mutation to fine tune a fit population created by selection and crossover, and that introducing mutation too early on in the GA would limit the effectiveness of crossover by weakening already fit members.

In all cases, I found setting the mutation rate to be something of a balancing act; we don't want the algorithm to distort too much of a good thing, but we want to prod it into considering possibilities it has not yet examined. In practice, I found the first technique effective throughout the algorithm with low mutation rates, and I found the third technique effective when applied later in the algorithm, since only the fittest members of the population were mutated in a "fine-tuning" manner. The third, however, provided higher fitness values.

5.1.4 *Replacement*

I implemented a variety of replacement mechanisms:

- *Replace Worst*, where the worst members of the population are weeded out (as in a typical GA).

- *Replace Random*, where random population members were replaced by the children if they had worse fitness values.
- *Replace Parents*, where the parents of the children are always replaced regardless of fitness.
- *Pick Best from Parents and Children*, where the best two of the four chromosomes live to the next generation. This method is more consistent with the Evolution Strategies [6].

In the end, the most critical factor was maintaining population variety, since the long chromosomes need an extremely varied population to find the best pieces of the optimal solution, and so *Pick Best* provided the highest fitness values.

5.2 Fitness Function

The fitness function is a combination of the following weighted values:

5.2.1 Total Coverage

Total coverage is calculated by summing the weighted coverage value of each employee staffed in each day-hour slot. Thus if an employee has a weighted coverage value of 15 and is staffed for 30 slots, this employee's contribution to the population member's fitness would be 450.

5.2.2 Average Coverage

Average coverage is the total coverage divided by the number of employees per day-hour slot. Higher values indicate more coverage per slot, on average.

5.2.3 Slot Distribution

As employees are assigned to day-hour slots, the algorithm keeps a separate data structure that stores a count of the slots each employee is assigned. This data structure is then flipped, creating a distribution grid of the number of times each staffing count is used (e.g. if 50 employees are each staffed to 10 slots and 25 employees are staffed to 5 slots, the distribution grid would store 50 at entry 10 and 25 at entry 5). For each distribution point we divide by the optimum distribution point (calculated by dividing the total slots staffed by the number of employees), and divide by the distribution point's distance from the optimum. This gives maximum points to more equal distribution of slots.

5.2.4 Employee Requested Slots Staffed

As employees are staffed to day-hour slots, we keep a running percentage of whether or not the employee requested the slot. This yields solutions that more fairly assign employees to slots they have requested.

5.2.5 Continuous Staffing

We calculate the percentage of staffed slots for each employee that is part of a continuous block of slots. This allows more continuous staffing to be favored.

5.2.6 Balance Factor

Finally, we use a balance factor to favor or disfavor equality among the competing factors. A higher balance factor yields more equally balanced solutions, although such balance means that the individual category fitness values will be below their optimums.

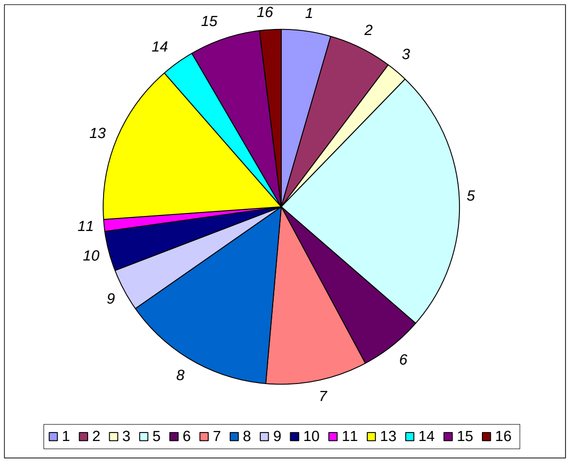
6. TEST DATA

Though based on real-life company data, as specified above the parameters are completely configurable and can be adjusted to any values necessary.

6.1 Employee Base

6.1.1 Coverage

I created and tested using several collections of 1000 employees that I believe to mimic the attributes of any real-life employee base, and particularly the employee base at the company. The following shows the average distribution of weighted coverage of the test employees:



As you can see, roughly half the employees have a coverage weight below 8 (only cover one of Math, English, Science, and Social Studies, or Math in conjunction with English or Social Studies), and the other half have a coverage weight of 8 or above (any other combination of coverage areas).

6.2 Administrative

6.2.1 Min/Max Desired Employees Staffed

I modeled the values after real-life company data, which closes its service from 1am to 12pm EST and requires a bump in staffing during peak (4pm to 9pm EST) hours:

Hours	**Min**	**Max**
12am to 1am	10	20
1am to 12pm	0	0
12pm to 4pm	70	100
4pm to 9pm	125	175

9pm to 12am	25	50

6.2.2 *Coverage Area Weights*

I used the following weights which correspond to the approximate value each subject-tutor provides:

Area	**Weight**
Math	5
English	2
Science	8
Social Studies	1

6.2.3 *Percentage of Coverage*

I used the following percentage of coverage values which mimic the real-world values:

Subject	**Coverage Percentage**
Math	61%
English	31%
Science	39%
Social Studies	20%

7. TRIAL RUNS

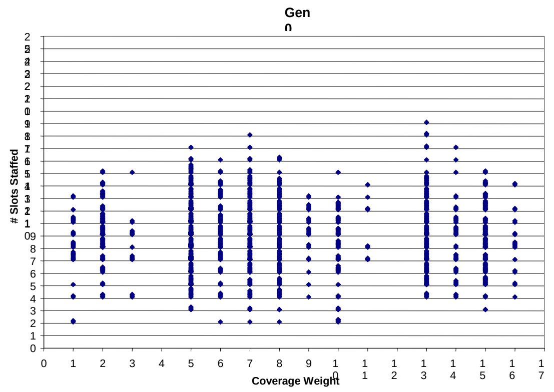
I averaged the results after 5,000 generations from several random populations, using several trial runs for each population in each of the following objective categories:

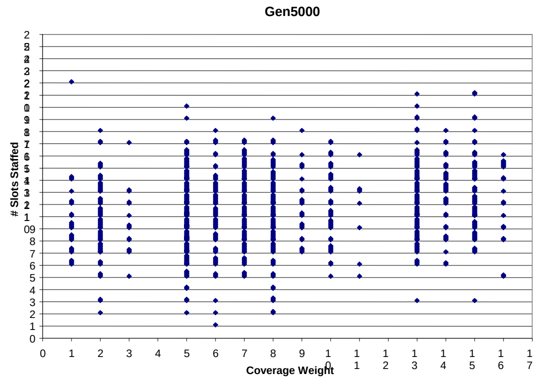
- *Coverage Only* (100% Coverage, 0% Distribution, 0% Requested, 0% Continuous)
- *Coverage+* (76% Coverage, 8% Distribution, 8% Requested, 8% Continuous)
- *Distribution+* (16% Coverage, 52% Distribution, 16% Requested, 16% Continuous)
- *Requested+* (16% Coverage, 16% Distribution, 52% Requested, 16% Continuous)
- *Continuous+* (16% Coverage, 16% Distribution, 16% Requested, 52% Continuous)
- *Balanced* (25% Coverage, 25% Distribution, 25% Requested, 25% Continuous)

With 1,000 employees and a population size of 500, the algorithm completes 5,000 generations in about 10 minutes on my 3GHz P4.

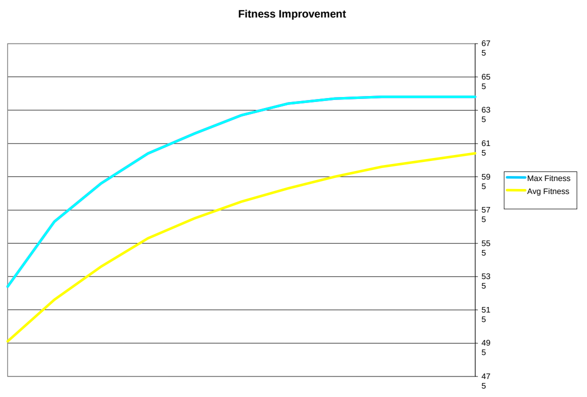
7.1 Coverage Only

To test the performance of the algorithm, I first experimented with maximizing only the *Coverage* objective. I figured that if working as expected, the algorithm would prefer solutions with employees with higher coverage values scheduled more. I considered the differences between the optimal population member at Generation 0 and the optimal member at Generation 5,000 in the number of slots given to employees of each coverage weight. Notice not only the overall bump up in total scheduling, but also the bump specifically for the employees who provide the best coverage. The average slots per employee with a coverage value of thirteen or higher is initially about 7, whereas at the algorithm's conclusion the average is closer to 12.





The following graphs show the improvement in fitness value and percentage gain from each generation over the previous:

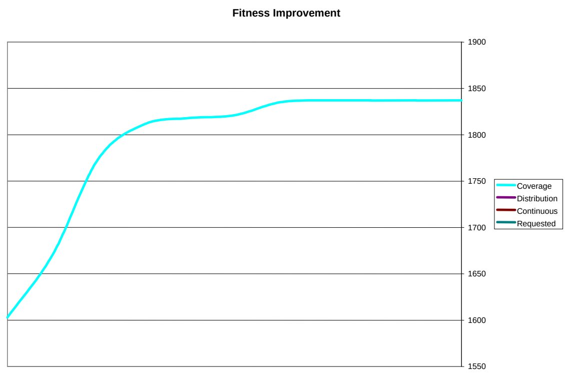


7.2 Single-Objective Optimization

By adjusting the objective weights, I created populations that favored one objective over the others. I then set the balance factor to 0 to allow the algorithm to maximize its best coverage candidate without suffering any balance penalty. This encourages the algorithm to optimize one category with less regard (though not *total* disregard) for the others.

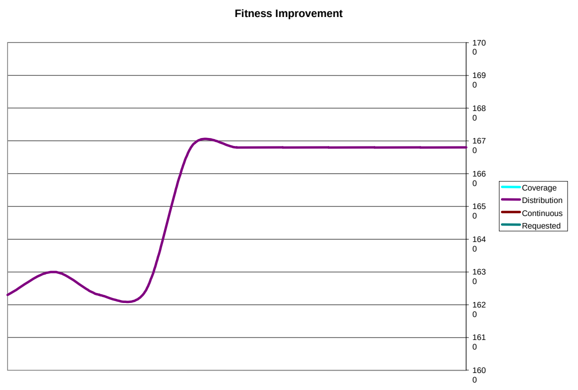
7.2.1 Coverage+

(76% Coverage, 8% Distribution, 8% Requested, 8% Continuous)



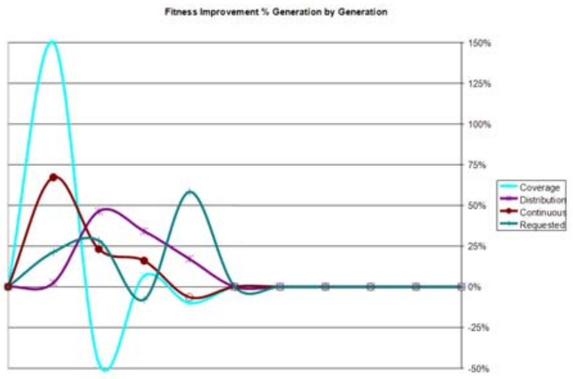
7.2.2 Distribution+

(16% Coverage, 52% Distribution, 16% Requested, 16% Continuous)



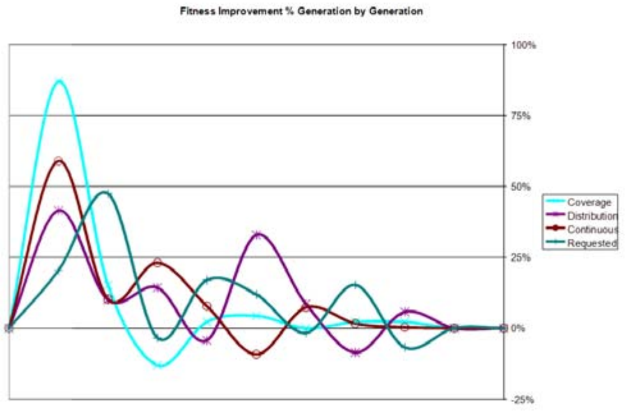
7.2.3 Requested+

(16% Coverage, 16% Distribution, 52% Requested, 16% Continuous)



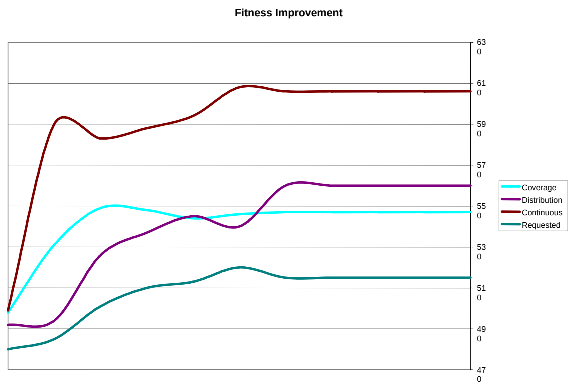
7.2.4 Continuous+

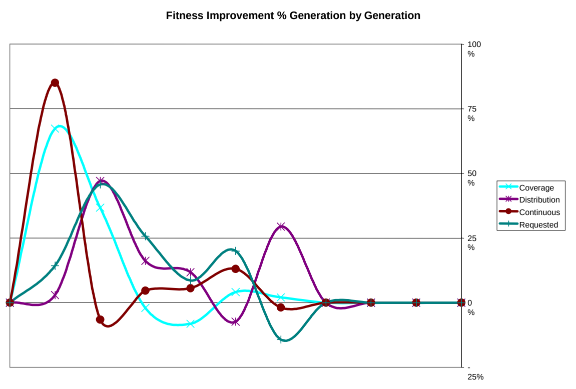
(16% Coverage, 16% Distribution, 16% Requested, 52% Continuous)



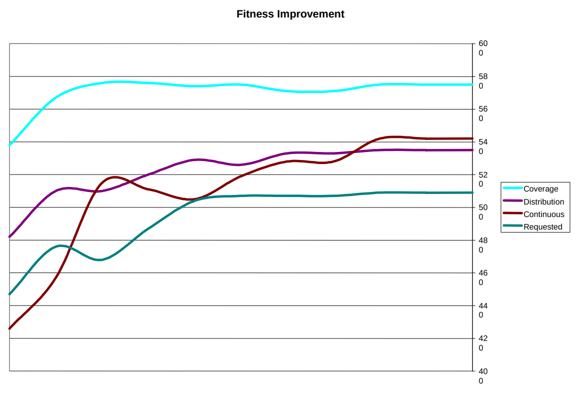
7.3 Balanced (25% Coverage, 25% Distribution, 25% Requested, 25% Continuous)

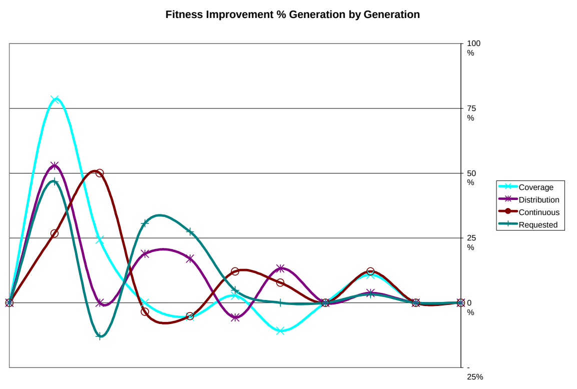
Three different result sets are presented below, each showing different patterns that the algorithm might take. These variations can be explained by the nature of the GA; although randomly generated, if the initial population is slightly slanted in one objective's direction, the final population will likely slant in that direction as well. The first shows one objective (*Continuous*) relatively maximized, two moderately maximized (*Coverage* and *Distribution*) and one with minor improvement:



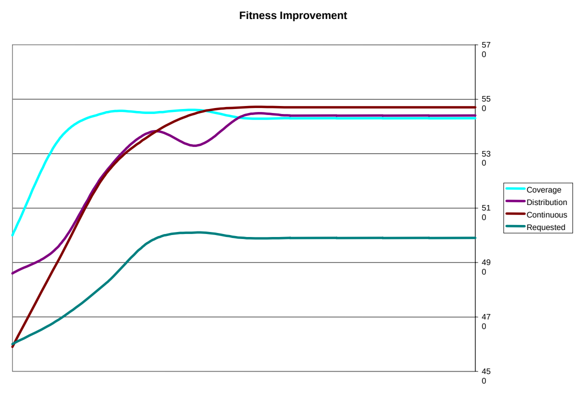


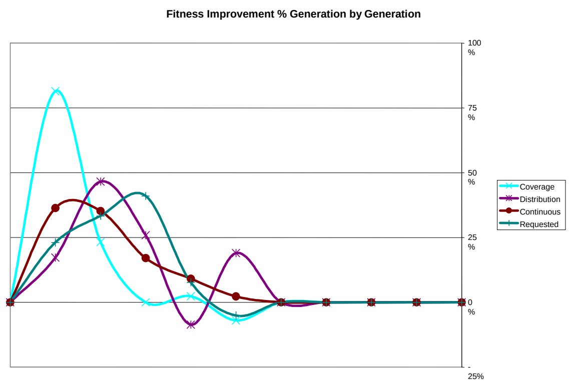
Next, I marginally bumped up the *Coverage* weight and so we see one objective (*Coverage*) leading the charge, with the other three close behind:





Finally, I set the weights almost completely evenly, and we see three objectives with impressive improvement and one (*Requested*) slightly behind:





8. CONCLUSIONS

The approach described herein builds on existing research and applies many of the characteristics of GAs and Evolutionary Strategies to a complicated problem space. Indirect encoding allows faster searching of the solution space and allows the genetic operators to go to work efficiently. With encouraging trial results, I believe that a GA that balances competing objectives is capable of handling the scheduling of tutors and other similar employee scheduling scenarios.

9. FUTURE WORK

An interesting avenue of further research would involve applying more of the statistical methods described in [11] and [12]. More well-defined queuing functions would improve the knowledge of the factors we are trying to optimize. Additionally, to account for differences in employee salaries, a future version of this algorithm should allow an administrator to specify min and max expense values and each employee’s hourly rate and the algorithm should take this overall cost into consideration.

10. REFERENCES

[1] Matthias Gröbner, Peter Wilke. Optimizing Employee Schedules by a Hybrid Genetic Algorithm, Lecture Notes in Computer Science, Volume 2037, Jan 2001, Page 463.

[2] Aickelin, U. and Dowsland, K. A. 2004. An indirect genetic algorithm for a nurse-scheduling problem, *Comput. Oper. Res.* 31, 5 (Apr. 2004), 761-778.

[3] Fukunaga, A., Hamilton, E., Fama, J., Andre, D., Matan, O., and Nourbakhsh, I. 2002. Staff scheduling for inbound call centers and customer contact centers. In *Eighteenth National Conference on Artificial intelligence* (Edmonton, Alberta, Canada, July 28 - August 01, 2002). R. Dechter, M. Kearns, and R. Sutton, Eds. American Association for Artificial Intelligence, Menlo Park, CA, 822-829.

[4] Shahid, A., Benten, M. S., and Sait, S. M. 1994. GSA: scheduling and allocation using genetic algorithm. In *Proceedings of the Conference on European Design Automation* (Grenoble, France, September 19 - 23, 1994). European Design Automation Conference. IEEE Computer Society Press, Los Alamitos, CA, 84-89.

[5] James C. Boyd, and John Savory. Genetic Algorithm for Scheduling of Laboratory Personnel. Clin Chem 2001 47: 118-123.

[6] Back, Thomas and Schwefel, Hans-Paul (1995). "Evolution Strategies I: Variants and their Computational Implementation." *Genetic Algorithms in Engineering and Computer Science,* edited by G. Winter, J. Periaux, M. Galan, and P. Cuesta, pp. 111-126. (c) 1995 John Wiley & Sons Limited. Used with permission.

[7] Jones, A. and Rabelo, J. C. (1998). Survey of Job Shop Scheduling Techniques, NISTIR, National Institute of Standards and Technology, Gaithersburg, MD.

[8] Philip Husbands. Genetic Algorithms for Scheduling. School of Cognitive and Computing Sciences, University of Sussex.

[9] Mehmet Kaya, Reda Alhajj. "Multi-Objective Genetic Algorithm Based Approach for Optimizing Fuzzy Sequential Patterns," *ictai*, pp. 396-400, 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'04), 2004.

[10] David S. Todd and Pratyush Sen. Multiple Criteria Scheduling using Genetic Algorithms in a Shipyard Environment, In *Proceedings of the 9th International Conference on Computer Applications in Shipbuilding*, Yokohama, Japan, October 1997.Dimensioning Large Call Centers

[11] G. Koole and A. Mandelbaum. Queueing Models of Call Centers. Industrial Engineering and Management, Technion, Haifa 32000, Israel, October 2001.

[12] Goldberg, D. E. 1989 *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. Addison-Wesley Longman Publishing Co., Inc.