Improved Crowd Control Utilizing a Distributed Genetic Algorithm

John Chaloupek

December 6, 2003

Abstract

Disasters are far worse when there's crowds of victims, as they are far more likely to get hurt or killed. In this paper we examine using a Genetic Algorithm to come up with a set of actions that first responders can take to reduce fatalities in this situation. We also look at using a Client/Server network model to make use of parallel computing resources. While we weren't able to get the Client/Server model fully working, our basic simulation of the crowd situation shows that Genetic Algorithms are effective for this purpose, even at such a basic level, and that the Client/Server network model would be an effective way to make use of additional computers.

Keywords

Genetic Algorithms, Evolutionary Algorithms, Crowd Control, Client/Server, Distributed Algorithms, Artificial Intelligence, Evolutionary Computation, Autonomous Agents

Contents

1 Introduction **2**
- 1.1 Motivation . . . 2
- 1.2 Background . . . 2
- 1.3 Research . . . 3

2 Design **3**
- 2.1 Genetic Algorithm Design . . . 4
- 2.2 Software Design . . . 6

3 Experimental Setup **7**

4 Results **7**

5 Conclusion **10**

6 Future Work **10**

7 Bibliography **11**

A User Manual **12**

B Fitness Plots with Standard Deviation **13**

1 Introduction

Evolutionary Algorithms have been shown to be useful at finding approximate solutions to many NP complete problems, due to their ability to selectively search the problem space and quickly converge on a "good" solution. Finding a set of actions first responders can take to control a crowd situation is one such problem.

1.1 Motivation

In today's world disasters such as Fires, Terrorist Attacks, Weapons of Mass Destruction, and Natural Disasters are not infrequent. And unfortunately, when they do happen they often affect large crowds of people. A bad situation, like a fire or a bomb, that would have been mainly a loss of property, if there were few or no people around when it happened, can become a lot worse as the panic and confusion of a panicked crowd of people claim more lives than the underlying disaster.

The first responders, such as Police, Fire Department, Hazardous Materials personnel, and others, need methods to respond to crowd situations. Unfortunately they have limited means to deal with crowds. The people that manage these situations need a way to evaluate the effectiveness of existing methods, as well as a way to test methods that aren't yet ready to be put into common use.

1.2 Background

Quite a bit of research has been done on modeling crowds as groups of intelligent agents in a virtual environment. [WJ94] defines what an agent is and shows us the different types of agents.

[KT] shows how agents can interact with each other and their environment.

[FMS+00] shows us a method for modeling a group of virtual human agents as a crowd.

These types of technologies are applied to virtual city modeling [TFB99].

And we're shown several ways to use Rules-Based approaches to defining crowd behavior [TMG99, SMGT99]

[MT00] gives us an overview of the problems commonly found when trying to model crowds.

1.3 Research

Even with all that research on how to model crowds, and give them complex behaviors, very little research has been done on the subject of controlling crowds. Thus I plan to discuss using a Genetic Algorithm that simulates what authorities might try to due to control a crowd situation during a disaster. This differs from the previous research in that the individuals in the crowd are following relatively simple "scripts" rather than having a complex set of goals and behaviors. The focus here is on trying to find what actions first responders could take to limit casualties in a disaster situation.

An Evolutionary Algorithm is a good choice for this type of simulation because EA's can help gather support for new methods that have yet to be proven effective. They can also lead to unexpected discoveries because they're somewhat random, and thus can examine solutions that most people would either not think of, or would dismiss because they seem like a bad idea, even though they might actually be effective.

To summarize, the goals of this project are:

- See if a system for simulating crowd behavior & crowd control using a Genetic Algorithm can be developed.
- Reduce (virtual) fatalities by suggesting sets of useful actions that first responders could take to manage a panicked crowd situation during a disaster.
- Examine the possibility of making this Genetic Algorithm distributed across a network of computers so that it can utilize more computing resources to achieve better results.

2 Design

Here I try to simulate the crowd situation by having a number of virtual agents, modeling the victims, run around the map of a simple building. There are sources of damage to simulate fires or other disasters. As the victims come into contact with these sources, they take damage, and if they take enough, they die. The first responders have set up a few precautionary measures, such as barricades, to help guide the victims out of the building. The simulation ends when all the victims have either escaped or died.

The types of victims are:

- **Panicker -** Runs around in a completely random manner.
- **Seeker -** Heads directly for the nearest exit.
- **Wimp -** Runs away from any source of damage/noise.
- **Follower -** Follows the nearest large group.

The actions the first responders can take:

Figure 1: The general outline of an EA. Adapted from Evolutionary Computation 1 [BFM00]

```
Population p
Set of Conditions c
Fitness Function f

Initialize(p)
Evaluate(p)

While(c != true)
{
    recombine(p)
    mutate(p)
    evaluate(p, f)
    compete(p)
}
```

- Place barricades
- Set up noise sources
- Direct people away from the scene

Due to time constraints it's been necessary to, at this time, set up a simplified system that only implements the Panicker victim type and the Barricade action.

2.1 Genetic Algorithm Design

A Genetic Algorithm is a type of Evolutionary Algorithm. Evolutionary Algorithms are stochastic algorithms that randomly create a population of individuals that are trial solutions to a problem that the EA is being applied to. These individuals are then evaluated according to some criteria that are believed to be an indication of how good of a solution they are to this problem. Then these individuals reproduce, mutate, and compete, see Figure 1 in a way similar to how members of a biological species do. Thus, the hope is that as more fit members reproduce more frequently than less fit members, an optimal, or at least very good solution to the problem will be found.

This is the type of Evolutionary Algorithm I've implemented. I've chosen to make selection and competition linear Rank Based, since it is a mature method that's been well researched, and allows the selective pressure to be easily changed to setting just one parameter. There is also an Elitist method so that the best solution is never lost.

Currently each individual is represented as a list of barriers, simulating police barricades, that can be placed on the map. Each barrier has an (x,y) position

Figure 2: Linear Ranking formula from Section 25.2 of Evolutionary Computation 1 [BFM00]

$$Pr_{rank}(i) = \frac{\alpha_{rank} + [rank(i)/(\mu - 1)]\,(\beta_{rank} - \alpha_{rank})}{\mu}$$

Where

μ is the population size.

β_{rank} is the expected number of offspring for the fittest member. It is the steepness parameter B specified in par.dat (see Appendix A: User Manual)

α_{rank} is the expected number of offspring for the least fit member. $\alpha_{rank} = 2 - \beta_{rank}$

$rank(i)$ is the rank of individual i. Where 0 is least fit and $\mu - 1$ is most fit.

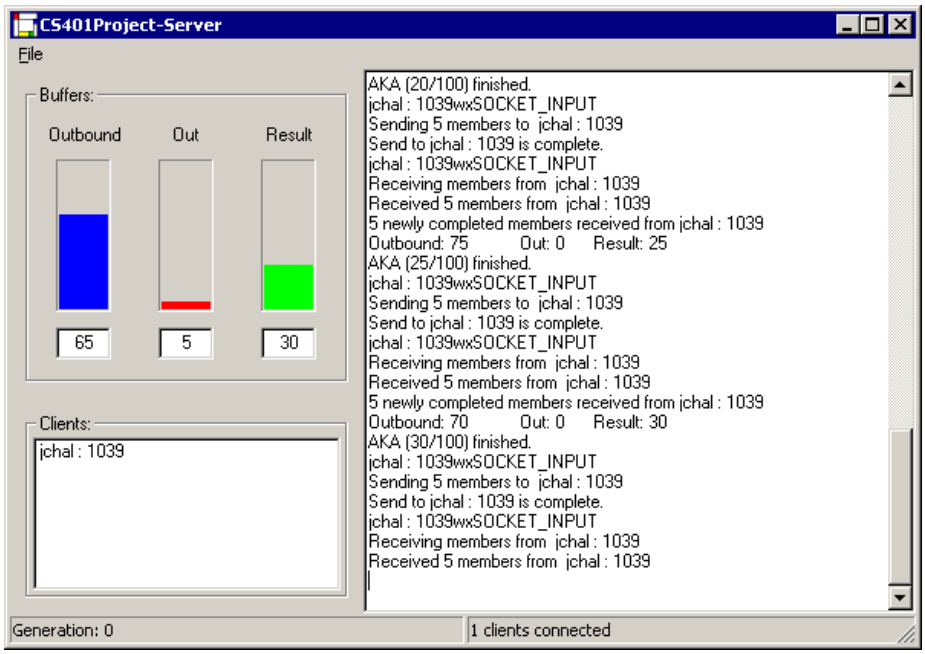
and an orientation (currently only vertical or horizontal). In the future, the individual may also contain a list of noise sources, each with an x,y position.

Crossover here is a 2-parent operation that is analogous to Uniform Crossover. Other problems, such as 3SAT or Binary Knapsack tend to have representations where a gene, usually the value of a variable, or whether a given item is in the knapsack, respectively, from one parent is mutually exclusive of the other. In other words, you can't have an item be both in and out of the knapsack. This is not the case with the list of barriers. So, each item in either parent's list has a 50% chance of getting passed on to the child. This makes it most analogous to Uniform Crossover, even though it's possible, though unlikely, that a child might have either all the barriers from both parents, or none from either.

Mutation happens as follows. If a member has no barriers in its list, it defaults to creating one and placing it randomly on the map. If it does have some in its list, it has a 1/3 chance each to Create a Barrier (as above), Modify a Barrier, or Delete a Barrier. Deleting a barrier just removes a random one from the members list. Modifying one means there's a 50% chance that either the orientation or the position change, but not both. If the orientation changes, it flips from vertical to horizontal or vice versa. If the position changes, it starts with the current position and adds a random number selected from a Gaussian distribution with a Standard Deviation of 5. This happens for both the x and y components of the position. The Standard Deviation is currently hard coded. There is only a check for whether barriers end up off the map, if so they're put back on the edge. No checks are made to see if barriers are in walls, or each other. This is a simplification, put in place for sake of brevity and speed, that may be removed in the future.

Fitness is determined by running the simulation. Then, when all the victims

Figure 3: Server Screenshot



have either exited the map, or have died, it takes an average of all their life and uses that for fitness. This evaluation takes into account both fatalities and overall health of the survivors. Since, optimally, it's better to get everyone out fine, rather than near death.

2.2 Software Design

The Client/Server model is the way that this system makes use of multiple computers. Most of the computational time is taken in the evaluation of members. This is a perfect opportunity to make use of parallel computing resources. To this end, the code was divided into two parts. The main GA runs on the server, reproducing members, starting new generations, parsing out members to clients, getting results back from clients, and logging them, starting the process all over again. A screenshot of the server window is shown in Figure 3.

This was programmed in wxWindows to gain the benefit of being cross platform. wxWindows works on Windows, MacOSX, and various flavors of Linux and Unix. Unfortunately, learning the intricacies of Client/Server networking on wxWindows was too much for the amount of time allocated to this project, thus it never worked quite right. The results in this paper had to be obtained from a simplified local-only version of the program. However, the need for a

distributed model still exists, and has been strengthened by the results from this simplified version.

The software design is very modular.

comdefs - Define a few Common Definitions that a number of classes find useful.

except - Provide some customized exceptions to be thrown.

ga - Provides the main Genetic Algorithm. Takes care of

- main loop (see Section 2.1)
- calculating statistics
- logging
- saving and loading of state

member - The representation of the members. When the algorithm is applied to a new problem this should be the only part that needs to change. Utilizes person, personderiv, and arena to run the simulations when the members are evaluated. Takes care of

- genetic operators (crossover and mutation)
- fitness evaluation

CS401ProjectEval - The main GUI class.

3 Experimental Setup

Nine parameter sets (see Table 1) were devised and then run, each 30 times so that a Z-test could be performed on the results (Table 2). The logs for each run have been saved and are available only in the soft copy of this paper, since they would add several hundred pages to this report. Each run logged the seed it used so that it might be rerun if there happens to be any doubt of its authenticity in the future. For each log period, the Current Generation, Best Fitness, Average Fitness, Standard Deviation, and Number of Distinct groups of fitness values, have been saved.

A group of base parameters (PS01) was established. Then, each variable was adjusted up or down separately. Also, one set (PS09) has both Probability of Mutation and Probability of Crossover set to 0 to establish a baseline for what could be expected before the GA aided the survival of the victims.

4 Results

Figure 4: Composite Graph contains graphs of the averages, of the best fitness from each run, of each parameter set. Appendix [ap] contains the same graphs, individually, but with their standard deviations graphed as error bars.

Table 1: Parameter Sets

Param Set	Bs	Bc	Pc	Pm	Pop
1	2.00	2.00	0.50	0.50	100
2	2.00	2.00	0.10	0.50	100
3	2.00	2.00	0.90	0.50	100
4	2.00	2.00	0.50	0.10	100
5	2.00	2.00	0.50	0.90	100
6	1.50	2.00	0.50	0.50	100
7	2.00	1.50	0.50	0.50	100
8	2.00	2.00	0.50	0.50	200
9	2.00	2.00	0.00	0.00	100

Table 2: Results

Param Set	Best Fit	Std Dev	Z Test	Run Time
1	30.0467	2.03219	2.524643999	4:52:30
2	30.14	2.13418	2.339563907	3:40:00
3	30.5783	1.71372	1.785219487	6:00:00
4	30.1783	2.11663	2.287608433	3:18:30
5	31.6683	2.87176	0	6:40:00
6	30.4567	2.41192	1.769537821	5:00:00
7	31.1967	2.81437	0.642407374	5:00:00
8	31.6633	2.45469	0.00724904	10:45:00
9	16.92	2.58756	20.89735697	0:25:00

Figure 4: Composite Graph

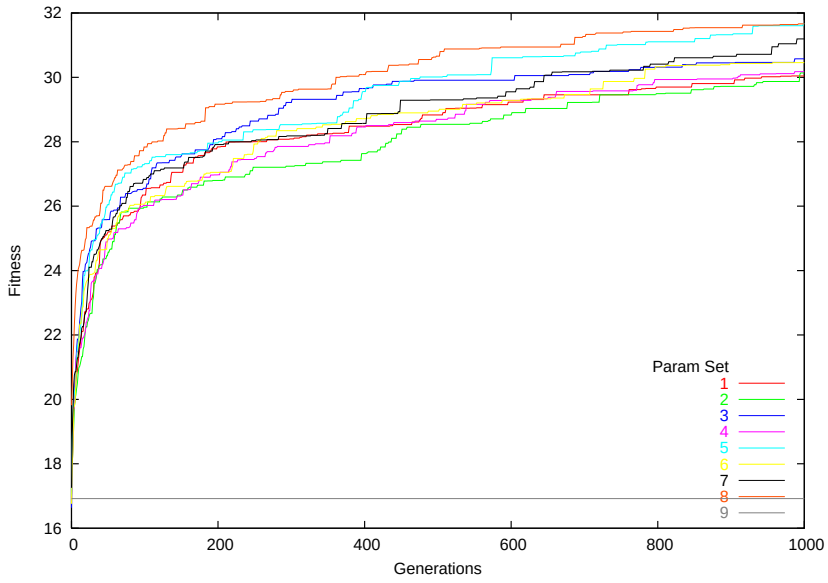


Table 2: Results shows the fitness at the end of the runs for each of the Parameter Sets. A Z-Test was performed to show which runs were significantly worse than the best, PS05. At the .05 certainly level they Z value has to be greater than 1.65, which it is for all except set PS07 and PS08.

One side effect of the PS09 baseline was that it ran far, far faster than the other runs. Approximately 7.94-25.8 times faster. This is due to that fact that no member had to be evaluated after the first generation, because there was no Crossover or Mutation, the only things that would change a member and cause re-evaluation. This clearly demonstrates just how much computational power the evaluations take, thus reinforcing the need for the Client/Server model.

5 Conclusion

We can thus conclude the following.

- learning wxWindows while doing network code AND setting up a Genetic Algorithm is hard.
- All the parameter sets are better than just random chance.
- Mutation plays a big role in getting good results for this problem. PS05 is the best parameter set because, even though it's statistically similar to PS08, it runs in about half the time.
- Most of the computation time is taken up during the evaluation of members, thus a Client/Server model that has a server running the GA, parsing out members to clients is an efficient way to make use of multiple computers for this problem.

6 Future Work

There can be much additional work done to improve this project. First there's the basics that have already been planned.

- Get network code running.
- Additional Actions
  - Set up noise sources
  - Direct people away from the scene
- Additional victim types.
  - **Seeker -** Heads directly for the nearest exit.
  - **Wimp -** Runs away from any source of damage/noise.
  - **Follower -** Follows the nearest large group.

Then there's work that can build on top of that.

- Let barriers be diagonal.
- Investigate if checking whether barriers intersect each other or walls is useful.
- Examine usefulness of varying Standard Deviation for barrier movement mutation.
- Run with more than 20 victims.
- More complicated buildings.
- Scaling: consider breaking problem down into parts (get out of room, then building, etc.)

7 Bibliography

References

[BFM00] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, Bristol, 2000. contains excerpts from the Handbook of Evolutionary Computation.

[FMS+00] Nathalie Farenc, Soraia Raupp Musse, Elsa Schweiss, Marcelo Kallmann, Olivier Aune, Ronan Boulic, and Daniel Thalmann. A paradigm for controlling virtual humans in urban environment simulations. *Applied Artificial Intelligence*, 14(1):69–91, 2000.

[KT] Marcelo Kallmann and Daniel Tallman. Modeling objects for interaction tasks. In *Computer Animation and Simulation '98*, pages 73–86.

[MT00] Soraia Raupp Musse and Daniel Thalmann. From one virtual actor to virtual crowds: Requirements and constraints. In Carles Sierra, Maria Gini, and Jeffrey S. Rosenschein, editors, *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 52–53, Barcelona, Catalonia, Spain, 2000. ACM Press.

[SMGT99] Elsa Schweiss, Soraja Raupp Musse, Fabien Garat, and Daniel Thalmann. An architecture to guide crowds using a rule-based behavior system. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 334–336, Seattle, WA, USA, 1999. ACM Press.

[TFB99] D. Thalmann, N. Farenc, and R. Boulic. Virtual human life simulation and database: Why and how, 1999.

[TMG99] D. Thalmann, S. Musse, and F. Garat. Guiding and interacting with virtual crowds in real-time, 1999.

[WJ94] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. HTTP://www.doc.mmu.ac.uk/STAFF/mike/ker95/ker95-html.h (Hypertext version of Knowledge Engineering Review paper), 1994.

A User Manual

The "par.dat" file takes one number per line, ignoring everything after that as a comment. The settings are as follows in the format (Line : datatype : description) :

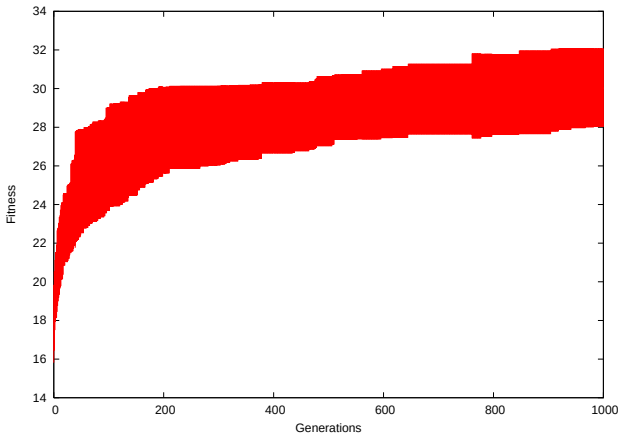
- Line 1: double: B-rank of selection: 1-2 (# of expected selections for best individual.)
- Line 2: double: B-rank of competition: 1-2 (# of expected selections for worst individual.)
- Line 3: double: crossover probability: 0-1
- Line 4: double: mutate probability: 0-1
- Line 5: int: population size
- Line 6: int: max generations to go.
- Line 7: int: log ever x generations.
- Line 8: uint: seed, 0 = choose from time.

First make sure your par.dat file is in the same directory with these settings. Also make sure you have a map.bmp file there. That file is a map of the building the simulation will take place in. The map.bmp that was used for the results in this paper has been included in the softcopy submission.

The map runs off colors:

- **black** is open ground.
- **white** is walls.
- **green** is the exit area.
- **red** is a damage source. Each point expands out to a radius of 10 pixels. If you put two points next to each other, they will each count as a separate damage source. They're pretty strong, so you should stick to point sources only in almost all situations.

Figure 5: PS1



Once you have those files, just double click the executable. The program will run and the status window will tell you that it's found and loaded the file (if everything goes well). Then choose "File — Run GA" and enjoy. A log file is generated named off the parameters in par.dat.

B Fitness Plots with Standard Deviation

Here are the Fitness vs. Generation plot of each individual parameter set, plot with error bars according to their Standard Deviation.

Figure 6: PS2

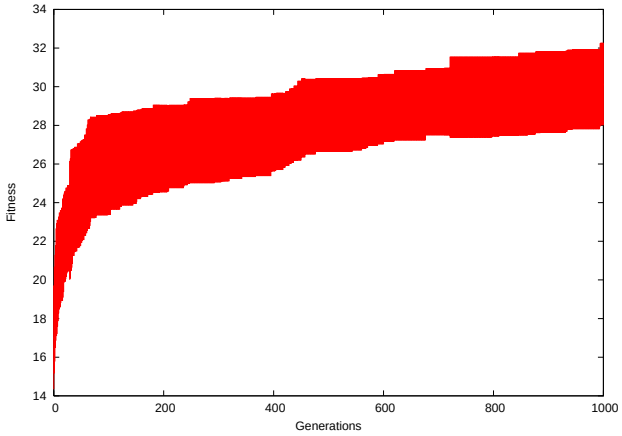


Figure 7: PS3

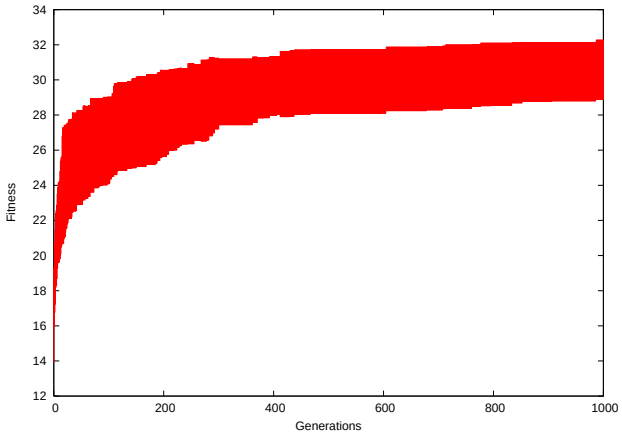


Figure 8: PS4

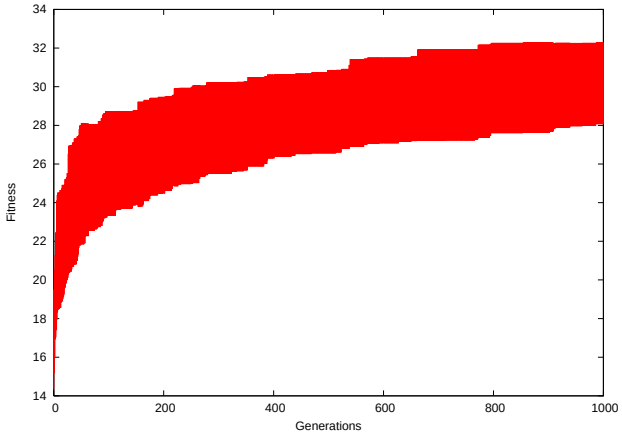


Figure 9: PS5

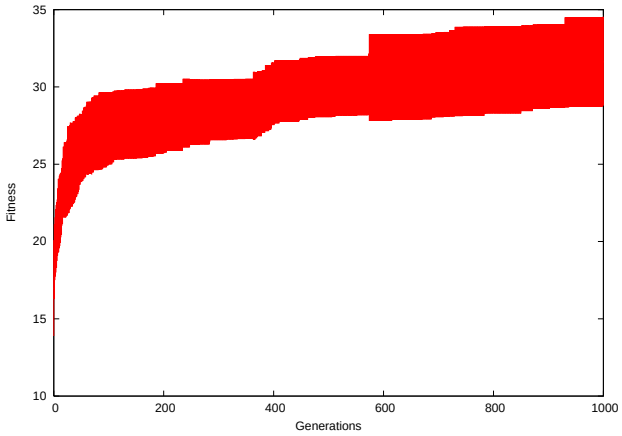


Figure 10: PS6

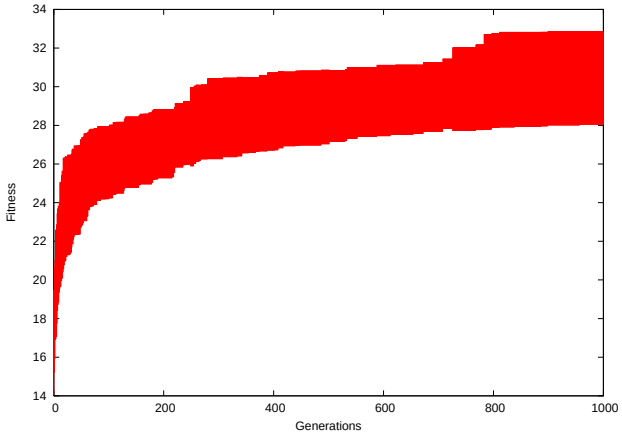


Figure 11: PS7

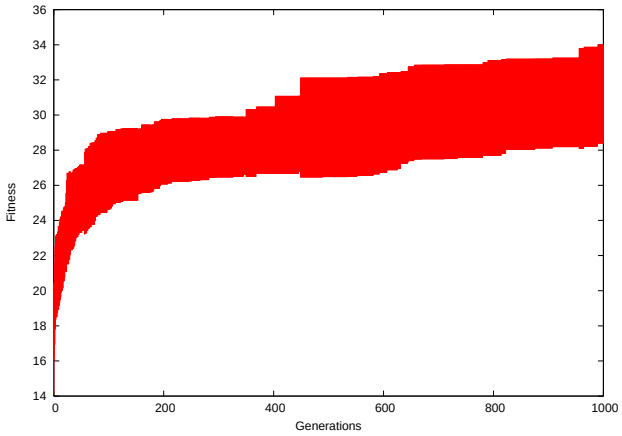


Figure 12: PS8

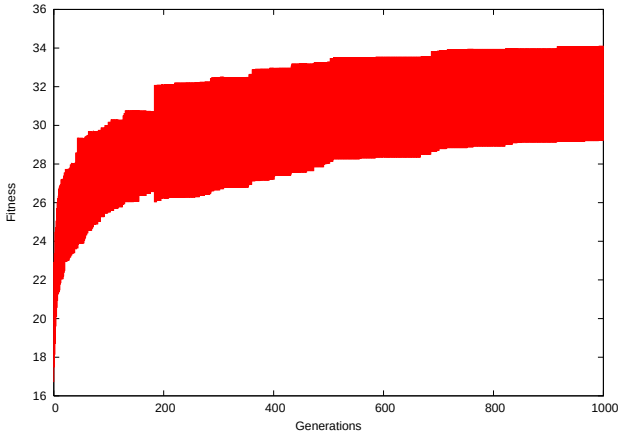


Figure 13: PS9