Applying Multiobjective Optimization to Path Finding

Gregory Timmons, Siddharth Sharma, Megha Umesha

I. Background and Previous Work

Genetic programming approaches have been successfully used for autonomous code optimization. Previous work has been successful in applying single objective genetic algorithms in reducing bugs in off the shelf c programs [1] and also for optimizing the run time of an existing MiniSAT solver implementations with reasonable results [2]. While they are generally not able to outperform an intelligent programmer, they are able to autonomously produce solutions that approach manually produced solutions giving us an avenue for automation of previously manual tasks.

Extending this to Multi-objective approaches to produce a Pareto optimal frontier have been applied weighing accuracy against running time for various estimation loops [3], for optimizing energy consumption against performance [4] and for producing Pareto optimal frontiers for a large set of objectives namely speed, size, throughput, power consumption and bandwidth [5].

All of this work is significant because it demonstrates the use of genetic optimization to a class of problems that is distinct from other commonly studied problems since the candidate solutions must be represented as a sequence of events rather than a vector of independent variables. This difference in solution space may have implications on the efficiency of various genetic programming techniques and it may not be true that what is optimal for the latter class is similarly optimal for the former in regards to our search strategy. This makes this an interesting area of study.

Another distinct feature of this class of problems is that they are very highly constrained. Finding valid code that compiles is not easy and this can be a limiting factor on the success of Genetic Programming.

Path finding falls into the same category of sequential solutions. A path is similar to a piece of code as both are lists of steps that must be taken by some agent. Genetic programming has also been successfully applied to path finding problems. NSGA-II, SPEA or similar algorithms has been successfully used to generate multi-objective optimized paths in a variety of contexts [6] [7] [8].

II. Introduction

In this project we will examine the application of multi-objective genetic algorithms to the realm of path finding and explore what factors might determine success. We are interested in unique qualities of the sequence based solution space which may be different than the traditional solution spaces examined by GA techniques. In particular we will explore two concepts taken from other work:

A. Smart Seeding

First we will explore the concept of selective seeding. In the paper by Andrea Arcuri et. al, they employ a concept they call smart seeding to improve the results of their search [9]. In the context of their paper smart seeding means giving as input for the initial genome a working version of a program that the code intends to optimize. This is a departure from typical GA applications where the initial input is purely random. We will argue that this initial seeding is greatly improves performance in a path finding context as well.

B. Path Representation

Second we will explore various methods of path representation and examine how they impact the performance of optimization. In the paper be Faez Ahmed et. al, they discuss 4 possible path representation schemes [7]. We will explore the third and fourth presented in that paper. The first we examine was a relative step notation that represented paths as a relative movement from the previous. The second version was an absolute path representation where each step represents an absolute location on the map to which the agent will move. Ahmed claimed that the second representation scheme was not suitable to this problem since it prevents the GA from optimizing subproblems, however we think it might be a viable option so we will explore it.

III. Path Finding

A. The Model

Maps in this project are represented as 2D grid with discrete locations and paths were scored via the rules of a very simple game. A path P_i is represented as a list of waypoints which the agent will travel to in succession. $P_i = (x_1, y_1), (x_2, y_2)...(x_n, y_n)$. Depending on the settings, this value can either be absolute point locations or relative locations. When traveling from one waypoint to the next, Bresenham's line drawing algorithm was used to calculate all of the points which the agent must travel through to complete the path. Each time an agent passes through a point in the map his score is adjusted according to the rule table below. A cell of type 0 is considered a wall any path that intersected a wall was thrown out. Rule e was evaluated for every point. In addition a constrain was placed on a path that health must never be less than or equal to 0, that is the agent must not die traveling the path. It was not a constraint that an agent must reach a goal point (5) however the rules of the game heavily weight an agent to want to reach the goal point.

Two metrics were computed not from the rules table based on the over paths and also included as objectives. These were distance from goal and exploration. The first is a measure of the maximum distance from the start that the path achieves and the second what percent of time the path spends on points it has not visited before. Both work to encourage paths which do not dwindle too long in a single spot.

Objective	Range	Goal
Steps	$[0, \infty]$	Minimize
Health	$[0, 100]$	Maximize
Gold	$[0, \infty]$	Maximize
Goal	$[0, 1]$	Maximize
Dist from Start	$[0, \infty]$	Maximize
Exploration	$[0, 1]$	Maximize

Objectives

Cell	Steps	Health	Gold	Goal
e	1	-1	0	0
1	0	0	5	0
2	0	-10	0	0
3	0	5	0	0
4	5	0	0	0
5	0	100	300	1

A typical rules table used

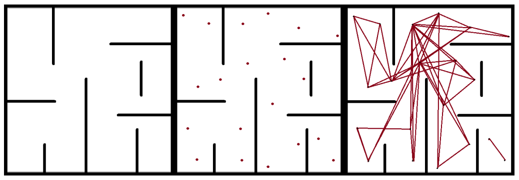
B. Initial Population Generation

The generation of the initial population was important for the success of our algorithm. Our experiments with unseeded populations gave underwhelming results. They would often fail to find even simple paths to the goal and required many generations. Therefor a lot of focus was put on an efficient and quick initial population generation technique. The goals for this technique were first to produce a set of path with very high variance and second to guarantee that as much of the space is covered in the initial population as possible.

A novel random path generation technique was implemented. First a randomized visibility graph was produced by randomly placing points with in the map and creating an adjacency list of traversable paths between points. Any point that intersected an obstacle or could not find any neighbors was thrown out. The number of randomized points was given as a parameter represented by a percentage of the total open space.

An initial population of size n are then generated by creating n paths with a single node at the start point and growing the paths round robin until every edge in the visibility graph is included in at least one path. To ensure that this does not take too long when selecting the next node in the path, the probability of selecting the next path is weighted inversely by how many times that path has already been chosen.

As part of our experiments we ran tests where the seeding was not guaranteed to cover the entire adjacency list. No weighting or checking for coverage was performed, instead path were chosen completely from random from the adjacency list. This will be referred to as "Unseeded" from this point onward.



Visibility Graph Example

A major shortcoming of this approach is illustrated in the example above. In this example there are 2 connected components. In this situation the path in the bottom right will never include these paths. To fix this a quick depth first search of the visibility graph checks for and unreachable points and will increase the number of nodes until all location have potential to be reached.

C. Genetic Optimization

As a baseline for performance a simple version of a MOGA was implemented. Due to the sequential nature of a solution for a path custom mutators had to be created. Crossover was implemented by taking a random prefix of one path and combining it with a random post fix from another path. Mutation was implemented to perform one of two operations with varying probabilities. First the mutator may chose to add a point randomly between any two other points in the path. Second any point in the path may be removed. In our implementation extra weight can be given to the probability that extra nodes are added to the end of a path should although we ultimately found this to be unimportant.

All fitness measurements were done using continuous domination. We found that when using binary domination to GA tended to favor short path and we would rarely get any meaningful results. Continuous domination handled the objectives given far better. While this was very interesting, and was ripe for exploration, we chose to limit ourselves to only continuous domination in this report to focus our research.

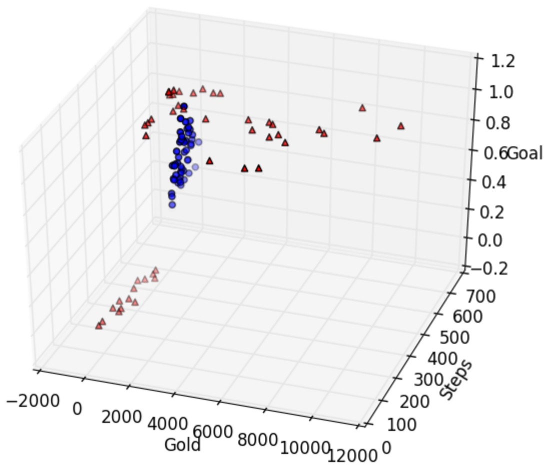
To assess the benefit of MOGA in this context another randomized path search algorithm was devised without a genetic component. This randomized search worked by repeatedly calling the initial population search function and simply keeping the best n paths by binary domination where n is the population size of the MOGA.

IV. Analysis

We found in general our algorithm had a very high degree of inter run variance. We were very prone to getting caught in local minima and the was a high degree of randomness to our results. In all of the following analysis number were recorded by running each of the approaches 50 times and the median path scores were reported for each run. We considered a few way to compare the algorithms since and decided on this since it captured the high amount of per run variance that we were seeing while still showing the overall distribution of performance.

A. MOGA vs Random Search

In the following figure the best solutions are plotted from both the randomized search and from the MOGA. The utopia point would be the top front left corner.

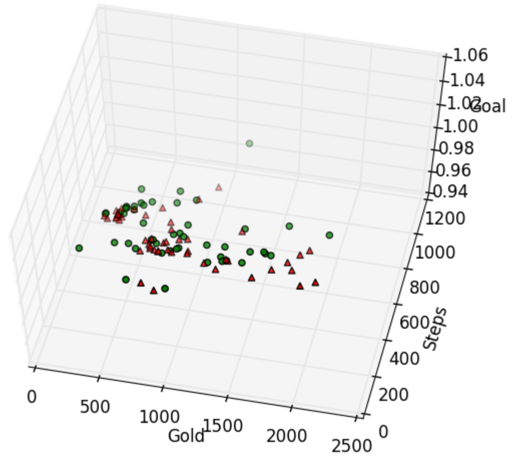


MOGA (Red) vs Randomized Path Search (Blue)

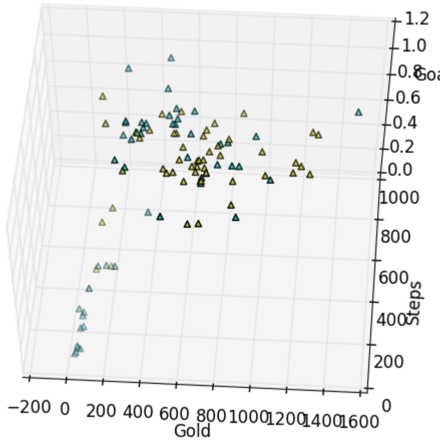
MOGA optimized paths are clearly outperforming randomized paths. Not a single random path dominates an optimized path and on top of that MOGA clocked in at 5.34 seconds while the randomized search took 91.34 seconds to run. MOGA is a clear winner here.

B. Seeded MOGA vs Unseeded MOGA

Here we compare seeded vs unseeded runs. We see that the smart seeding efforts improve the quality of the paths. The first graph is for absolute pathing while the second is for relative pathing. In both cases smart seeding give definitely improved results – how ever the improvement was a little more drastic in the Absolute category.



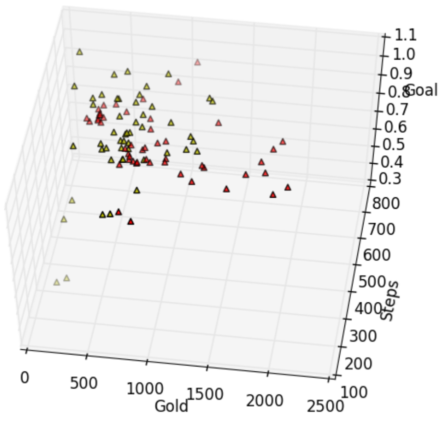
Seeded Absolute(Red) vs Unseeded Absolute(Green)



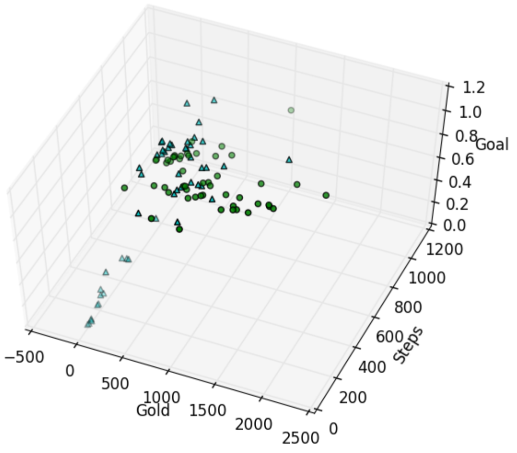
Seeded Relative(Yellow) vs Unseeded Relative(Cyan)

C. Absolute Paths vs Relative Paths

Next is a break down of absolute vs relative path representation – first we look at unseeded examples and after we look at seeded examples. Here we see contrary to claims made in the Ahmed's work [7] our results suggest that absolute space representations are suited to Genetic Algorithms as our absolute representation seem to out perform our relative implementation.

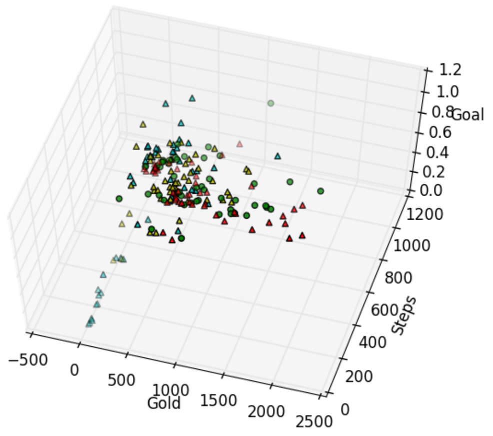


Seeded Absolute (Red) vs Seeded Relative (Yellow)



Unseeded Absolute(Green)vs Unseeded Relative(Cyan)

All results on one plot :



Seeded Absolute (Red)
Seeded Relative (Yellow)
Unseeded Absolute (Green)
Unseeded Relative (Cyan)

D. IGD Comparisons

V. Conclusion And Reflection

In this project we found that selective seeding and absolute path representations are good approaches for improving GA applied to path finding problems. Both gave modest improvements in the expected optimality of our algorithm.

However despite our best efforts our implementation failed to deliver what we had hoped for. At the onset of this project we wanted to be able to produce a set of highly varied and optimal paths which would have given the user a set of options for navigating the map. What we discovered, early in the project was that a general path would usually quickly dominate the population and the population would soon become variations of similar paths.

In broad strokes our algorithm would first find some decently good solution or path, then iterate over the path to find more optimal version of this general path. Worse – sometimes that "decently good" solution wasn't that good at all, leading to the algorithm basically failing to produce a reasonable path at all.

What we observed was that over time the "gene pool" of our populations becomes distinctly less diverse. While this is not all bad since we want "bad genes" to fall out - it falls short of our hope for producing multiple solutions as diversity tended to fall too far. This seems to be the curse of the local optima in our problem. We have decided on a few possible reasons for this.

First, we believe the sequential nature of the solution increases the potential search space, making search more difficult. In a typical problem with n variables we should expect that the actual solution space would have at most n or fewer real dimensions that must be searched. In our path finding problem the length of paths is unbounded. Each extra step increases the number of dimensions that needs to be searched. The longer our paths get, the higher the dimensionality becomes and the harder search becomes. We suspect there is a real upper limit to the practicality of this algorithm as paths become longer. Paths are not kept to similar lengths. It might be possible that a shorter paths are better able to search their local space for optimizations than longer paths since the effective search space of a shorter path is smaller. When competing for space within the population, this may have given shorter paths an advantage.

Another possible reason for the lack of variety in general solutions might have been that GA also tends to struggle when there are too many constraints and we believe that the map like environment that we were working with was highly constrained. Walls and death made finding valid solutions hard. This prevented us from coming up with mutators sufficient to search very far from the current solution. When moving a single part of the path a few points we can often find a valid solution, but with a complex map randomly mutating a solution to search a new room, or to go around a object a different way is very improbable. This meant our mutations tended to only make small changes to existing paths.

A similar shortcoming existed with out crossover. Simple crossover will select from either the mother or father on a single dimension. Our crossover permutes the genes. This means if in a population of 100 if there are 100 distinct values for a given point, simple crossover can select any of those 100 values. Conversely, our crossover can potentially select any value in any dimension to place at that point – potentially far more than 100 values. It seems that this large space in tandem with the highly constrained solution space played a role in favoring lower variety, since with a more random crossover in a very constrained environment find valid solutions is unlikely.

A quick analysis of the mutation numbers showed that crossover in valid paths tends to always strongly favor a single parent. Crossovers that combined both a father and mother evenly rarely survived. To make a biological analogy: the only children that survived the mutation process were generally the children of inbreeding which of course had a negative effect on genetic diversity.

Thus we came to the conclusion : Inbreeding kills our algorithm. Paths are strongly encouraged to crossover with themselves and mutations rarely are able to create entirely new paths. Slowly other general paths die out and the population becomes dominated by a set of genetically similar paths. At this point there is not enough in the mutation to push the paths out and the algorithm will sit in this local optima indefinitely.

We think that selective seeding was successful because it injected the gene-pool with a lot of variety – which improved our chances of picking the best local optima from the start. We think that absolute path points was successful because it shrank the the search space which helped the mutators and crossover in the highly constrained environment. Neither were enough to give us the result we were really looking for.

VI. Future Work

We observed a lot of issues with using Genetic Algorithms for path finding. On of the most obvious next steps would be to attempt to use SPEA-2 to try to improve the inbreeding problem. This seems to be exactly what SPEA-2 is meant to address so it seems like it would apply well. By selecting against solutions that are similar we would help to keep a more diverse gene-pool and improve our search.

We could also try to improve our mutation and crossover strategy so that we are better able to overcome local optima but this would probably come second to implementing SPEA-2 which has proven results in literature.

Another possible extension is NSGA-II whose frontier based sort may also help to prevent loss of diversity in the gene pool.

Throughout this work it has also become clear that objectively measuring and comparing the results of algorithms like this is not a trivial task. In future work we need to refine our ability to measure and compare various algorithms and we would hope to develop a good understanding of how to tell when a multi-objective path finder is working well. Because of time constraints

References

[1] S. Forrest, T. Nguyen, W. Weimer, and C. Le Goues, “A genetic programming approach to automated software repair,” New York, NY, USA, pp. 947–954, 2009. [Online]. Available: http://doi.acm.org/10.1145/1569901.1570031

[2] J. Petke, W. B. Langdon, and M. Harman, “Applying genetic improvement to minisat,” in *Proceedings of the 5th International Symposium on Search Based Software Engineering - Volume 8084*, ser. SSBSE 2013. New York, NY, USA: Springer-Verlag New York, Inc., 2013, pp. 257–262. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-39742-4_21

[3] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, “Managing performance vs. accuracy trade-offs with loop perforation,” in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE ’11. New York, NY, USA: ACM, 2011, pp. 124–134. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025133

[4] B. R. Bruce, J. Petke, and M. Harman, “Reducing energy consumption using genetic improvement,” in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO ’15. New York, NY, USA: ACM, 2015, pp. 1327–1334. [Online]. Available: http://doi.acm.org/10.1145/2739480.2754752

[5] M. Harman, W. B. Langdon, Y. Jia, D. R. White, A. Arcuri, and J. A. Clark, “The gismoe challenge: Constructing the pareto program surface using genetic programming to find better programs (keynote paper),” in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2012. New York, NY, USA: ACM, 2012, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/2351676.2351678

[6] M. Davoodi, F. Panahi, A. Mohades, and S. N. Hashemi, “Multi-objective path planning in discrete space,” *Appl. Soft Comput.*, vol. 13, no. 1, pp. 709–720, Jan. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.asoc.2012.07.023

[7] F. Ahmed and K. Deb, “Multi-objective optimal path planning using elitist non-dominated sorting genetic algorithms,” *Soft Comput.*, vol. 17, no. 7, pp. 1283–1299, Jul. 2013. [Online]. Available: http://dx.doi.org/10.1007/s00500-012-0964-8

[8] A. Lavin, “A pareto front-based multiobjective path planning algorithm,” *CoRR*, vol. abs/1505.05947, 2015. [Online]. Available: http://arxiv.org/abs/1505.05947

[9] A. Arcuri, D. R. White, J. Clark, and X. Yao, *Multi-objective Improvement of Software Using Co-evolution and Smart Seeding*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 61–70. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89694-4_7