A Genetic Algorithm Using Changing Environments for Pathfinding in Continuous Domains

Jared D. Mowery
Department of Computer Science
Stanford, California 94305
jmowery@stanford.edu

Abstract: This paper describes a technique for finding paths through a two dimensional continuous space using a genetic algorithm. While A* currently dominates the field of pathfinding, it requires that the space be discretized. For spaces which are not easily discretized, or in which a fixed grid size is not permissible, a pathfinding algorithm which does not require discretization is necessary. In addition to not requiring the discretization of the search space, a genetic algorithm employs a depth-first strategy which may yield suitable paths early in the search. This algorithm dynamically morphs the search space from an easy space into a difficult space to reduce the number of generations required to find a solution to the original space. The algorithm is demonstrated on a small set of sample environments consisting of polygonal obstacles.

I. Introduction

Pathfinding is a central problem in many endeavors. Current pathfinding is dominated by discretization of a space, if needed, followed by application of the A* algorithm. This is augmented by a variety of abstraction techniques which reduce the size of the search space to speed the pathfinding process. However, pathfinding in continuous spaces remains a difficult problem. While genetic algorithms may not seem appropriate for pathfinding, with sufficient refinement, they are useful. In particular, a genetic algorithm can quickly find a path through a continuous space without requiring discretization.

Since genetic algorithms are not complete, the pathfinding algorithm may be best used in combination with an A* search. If the genetic algorithm fails to find a path quickly, the A* algorithm may be used to exhaustively search the space. The faster the genetic algorithm can find a path, the lower the average pathfinding time becomes. The genetic pathfinder should also be useful in any cases in which the space cannot be discretized. While mathematical approaches exist for such spaces, it is much simpler, and, with sufficient research likely faster, to use a genetic algorithm.

Section II describes the problem which the genetic algorithm solves. Section III describes the problem specification in depth. Section IV discusses the operators used. Section V discusses the changing environment solution. Section VI reports results. Section VII concludes.

II. Statement of the Problem

The problem is to find a path through a continuous, two dimensional space filled with polygonal obstacles which does not intersect any of those obstacles. Since this algorithm is designed to be applicable to arbitrary obstacles, the specifics of the polygons cannot be used to speed the search. The path does not need to be optimal, as in A*.

III. Specification

The successful application of genetic algorithms to any problem relies heavily on the representation of the problem and the implementation of the genetic operators. The following is a tableau which summarizes the specifics of the algorithm:

Tableau:

Objective: To find a path through a series of polygonal obstacles using a genetic algorithm.
Representation Scheme:

Structure = Each element of the representation is a pair of floating point values defining an (x, y) point in the space.

K = infinite (alphabet of representation, all pairs of floating point numbers in bounds)

L = 2-10 (length of representation, anywhere from 2 to 10 points.)

Mapping from points in the search space of the problem to structures in the population: The representation of each member of the population identically maps to a point in the search space, that is, a path.

Fitness cases: Whether the path intersects a line segment of an obstacle.

Fitness: a linear combination of the number of intersections and the Manhattan distance of each line segment.

Parameters:

Population size M = 50
Intersection divisor = 0.1
Path length divisor = 100.0
Morph divisor = 50 default, variable otherwise
Crossover probability = variable
Mutation probability = variable
Insertion mutation probability = variable
Knowledge mutation probability = variable
Knowledge mutation maximum = 10
Knowledge mutation maximum add = 5
Mutation distance = 30.0
Maximum number of generations = 1000

Termination Criteria: a complete path with no intersections is found or the maximum number of generations is exceeded.

Result Designation: the best path.

Representation Scheme

A natural representation of a path is a set of pairs of real numbers, each denoting a point along the path. The path consists of line segments connecting those points. Therefore, the first and last points must be the starting position and the destination. The remaining points, hereafter referred to as **active points**, can be changed while still providing a working path. Since the space may be filled with an arbitrary number of obstacles, a variable length genetic algorithm must be used to allow the path to grow in complexity as needed to avoid the obstacles. For simplicity, the path length is limited to ten, an ample amount for the sample space. Since each point is a pair of real numbers, disregarding the precision of double values, the number of such points is infinite. Each path is represented by the points which compose it, such that each path corresponds directly to a path in the set of all possible paths, the search space.

Fitness

There are a variety of options for determining the fitness of a path. Two obvious variables are the length of the path and the number of intersections of the path with obstacles. The length of the path may be represented using the Manhattan distance rather than the true distances, since minimizing the Manhattan distance is equivalent to minimizing the real distance. This carries the advantage that computing the Manhattan distance is much faster than computing the real distance using the Pythagorean Theorem. The number of intersections of the path with obstacles is a more complex fitness measure. The measure must contain enough information about the intersections to allow the genetic algorithm to distinguish the better paths from the poorer paths, and yet be sufficiently simple to be applicable to arbitrary obstacles and efficient to compute. One good measure, and the measure used in the algorithm, is the number of intersections with each obstacle. Using efficient algorithms to compute the intersection of each line segment of the path with the line segments comprising the obstacles yields a fast measure. A better measure would be based on the length of the path inside of obstacles, but this is expensive and could require complex calculations even for relatively simple obstacles. Dynamic programming may make such a measure fast enough for practical use. A discussion of possible advantages of such a measure appears in the future research section. The fitness value is computed by dividing the number of intersections by the intersection divisor parameter and adding that value to the result of dividing the Manhattan distance path length by the path length divisor. The fitness value determined is normalized to a range defined using the highest fitness value of any member of the

population and the lowest such value, such that one half of that range is used. This encourages survival of unfit genes, which helps encourage diversity in the population--an important factor for pathfinding.

Parameters

The population size is the most important parameter. It should be sufficiently large to allow the genetic algorithm breadth to search the areas in the search space most likely to yield a path. However, too large a population simply wastes computational time with redundant paths. The population size will depend on the problem at hand, and is difficult to guess. The algorithm uses a population size of fifty, which is large enough for the sample problem space. A technique for dynamically changing the population size as the algorithm runs is discussed in the future research section.

The intersection divisor determines how much weight the critic gives to paths which avoid intersections. The lower the value, the greater the weight. The path length divisor serves the same purpose, except for path length. The combination of values presented makes avoiding intersections paramount, with reducing path length a comparatively minor concern.

The morph divisor controls the operation of the adaptation approach, described in Section V. The higher the divisor, the faster the obstacles are morphed.

The Mutation distance parameter controls the maximum x or y distance of a mutated point from an initial point position. This parameter is the same for the uniform probability distribution used to generate points in the kMutation operator, discussed in Section IV.

The remaining parameters regard the probability of the genetic operators being applied to a given member of the population. The probabilities are handled using a Roulette wheel algorithm similar to that discussed regarding reproduction. The operators named are described in Section IV.

Termination Criteria

The algorithm terminates when a path which intersects none of the obstacles is found, or after the maximum number of generations has been exceeded. The maximum number of generations is an application-dependent variable. The algorithm uses a value large enough to allow most searches to finish successfully. However, there are applications, such as the combination of the A* search with a preliminary genetic pathfinder, which would benefit from a lower maximum number of generations. The algorithm terminates as soon as a path has been found since specialized smoothing techniques can be used to refine the path more rapidly than the genetic algorithm.

Result Designation

The result is the first path which connects the starting point to the destination point without intersecting any obstacles.

IV. Methods

This section describes the operators used to solve the problem described above.

Crossover

A naive crossover operation would choose some number of active points at the end of a path to cross with another path. However, this would only allow the crossover operation to benefit the later portion of the path. This algorithm uses an implementation which allows the crossover operation to take any contiguous subset of the active points defining a path for crossover.

Mutation

The mutation operator, in simplest form, could replace any active point with a random point within the space. While this conceivably allows the genetic algorithm to construct any path, and to recover any lost path, it may not be optimal.

In particular, the mutation operation can be extended to use some knowledge about the domain to aid in performing knowledge based mutation. The knowledge based mutation operator, hereafter referred to as the **kMutation operator**, examines each line segment in the path, up to the maximum bound specified by the "knowledge mutation maximum" parameter. For each, if that line segment intersects at least one obstacle, it computes a representative midpoint of the intersections along that line segment. This point is constructed by summing each intersection points x or y component and then dividing by the number of intersections. A new point

for insertion into the path is then generated according to a uniform distribution within a bounding box around that representative midpoint. This point may be inserted into the path or replace a nearby point in the path, depending on whether the "knowledge mutation maximum add" parameter has been exceeded by the current path length.

The kMutation operator allows the path to mutate in small steps, such that a good path may be altered in the hopes of avoiding an obstacle. If the offspring does not avoid the obstacle, little harm is done. If the offspring successfully avoids the obstacle, the reproduction operator will favor it in later generations. Thus, repeated application of this knowledge based mutation operator can generate a complete path from start to finish. In fact, applying this operator to every gene in a size ten gene pool is sufficient to greatly outperform a standard genetic algorithm consisting of a naive mutation operator and the crossover operator described above.

A variant of the kMutation operator which changes only the first line segment containing an intersection is used in some experiments involving the adaptation process. When this version is used, it will be noted explicitly.

Reproduction

Reproduction is implemented in a Roulette wheel style algorithm. Each population member's fitness is used to determine a portion of the wheel. Then, a random point on the wheel is chosen. If that point falls within a population member's portion of the wheel, that member of the population is reproduced. Hence, a good member of the population may have several offspring in the next generation. Also, there is no safe guard to protect the fittest individual of the population.

V. Adaptation

This section describes an unusual approach to solving the problem. Conceptually, the problem of finding a path is converted from a problem in which all obstacles are static into one in which all obstacles are dynamic through the generations. The first generation has no obstacles to avoid. From that point forward, the obstacles grow slowly over time until they reach their full extent by a generation chosen as some fraction of the maximum number of allowed generations. For polygons, the polygon present at generation i is represented by a polygon whose vertices are a linear interpolation between a point in the center of the polygon and the final vertex positions.

This approach allows members of the population to more naturally adapt over time. Rather than requiring the genetic algorithm to find a perfect solution from scratch, it can find a reasonable solution and incrementally improve that solution over time. This process is evident in nature. When creatures first made the transition from sea-based life to land-based life, they did not form a perfectly adapted land-based individual in the sea to send forth. Rather, hybrids of sea and land based life evolved over time. As these hybrids became increasingly adapted to life on land, they spent less time in the sea. Finally, they began to dwell solely on land.

The following series of figures illustrate the process of polygons growing from points to their full extent. Figure 1a shows the polygon at generation zero. The gray lines denote line segments from the center of the polygon to the final vertex positions. The point chosen as the center of the polygon may be the centroid or any point generally in the center of the polygon. Arbitrary points could be used and the genetic algorithm would still search successfully given enough time, but points in the center of the polygon make the most sense, as will be clear later. Currently, the polygon is a point. Figure 1b shows the polygon at an intermediate generation. It has grown to become a small image of the final polygon. Figure 1c shows the polygon in its final form: it is completely extended. Any solutions found at this point are solutions to the original problem.

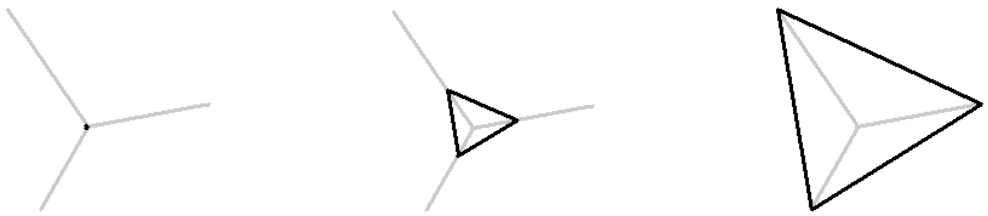

Figure 1. The figures are a, b, and c from left to right.

Recreating the pathfinding problem in a dynamic environment raises several questions. First, is there any benefit to be derived? Second, what operators are appropriate for this new problem?

Several factors suggest that this approach may help find solutions to a wide variety of problems more quickly. Any problem which can be decomposed into a spectrum of problems of increasing difficulty is a candidate for this technique. By slowly increasing the difficultly of the problem, the dynamic environment allows the genetic algorithm to solve a series of easy problems rather than one large problem. Furthermore, the dynamic environment can allow the use of techniques which would not be possible in a static environment, or, by reducing the difficulty of the problem, allow otherwise less effective techniques to achieve greater performance. In particular, the kMutation operator can be more effective, because it incrementally changes paths by relatively small amounts. The amount by which the kMutation operator is allowed to change the path can be smaller than in the static case because the static case requires guessing an improved path which may have to detour by an arbitrary amount. In contrast, the dynamic environment makes it more likely that the path once worked and now does not, such that the maximum change in the path required is directly related to the speed at which the polygons increase in size. The dynamic environment may also make it easier to solve problems involving closely spaced obstacles.

The series of figures 2a, 2b, and 2c show a hypothetical expansion of a polygon combined with the alterations made by the kMutation operator to a single path, to illustrate the process and the potential benefit of the dynamic environment.

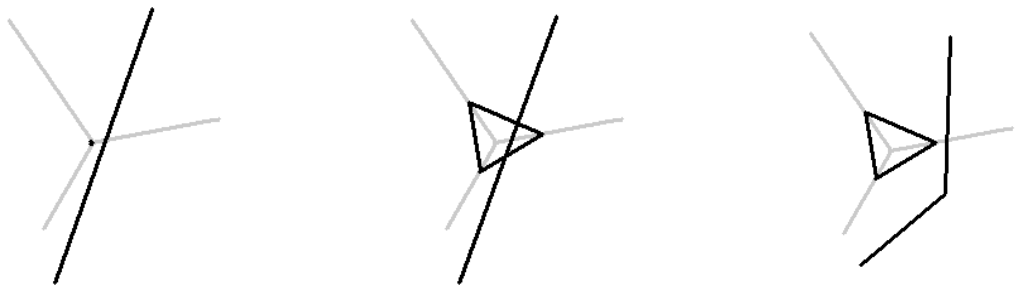

Figure 2a, left: The polygon is a single point, and the path does not intersect it.
Figure 2b, middle: The polygon has expanded to about one third of its full size, and now intersects the path.
Figure 2c, right: The kMutation operator has adjusted the path to avoid the small obstacle.

The kMutation operator serves as an example of an operator which may be appropriate for this dynamic space. Another obvious candidate is the dominance operator used with diploidy to allow abeyance. However, while the space is now changing over time, the polygons never contract. As such, no solution which worked in the past and then ceased to work will work in the future. Since this is likely the primary advantage of using these techniques, they are unlikely to be applicable to this type of search. (Goldberg 148-161) Other possible operators are discussed in the future research section.

Results

This section presents data regarding the performance of a number of variations of the genetic algorithm for the three samples spaces shown below for a variety of implementations. The spaces are referred to as 1, 2, and 3, respectively. The dots at the left and right of each figure indicate the points between which a path needs to be found.

Figure 3: the sample spaces used.

The kMutation operator, not surprisingly, surpassed both the standard mutation operator and the insertion mutation operator. In fact, the kMutation operator performs better than crossover: the first problem is solved by generation 32 when the kMutation operator is applied 50% of the time and the crossover operation is never applied. In contrast, when each is applied 25% of the time, the average generation of success is only 74. The kMutation operator provides diminishing returns in terms of reducing the average generation of success as it is applied more frequently, representing the relatively reduced operation of reproduction. This performance is vastly greater than a benchmark genetic search algorithm using a mutation operator which chooses a point at random in the space to insert or replace an existing point in a path.

Using the adaptation approach described in Section V the time required for the search is greatly reduced. When using adaptation, experiments which tested the effectiveness of different probabilities of the crossover and kMutation operations showed (the sum of both probabilities being fixed at 50%) that assigning a probability of between 0 and 5% for crossover and between 45 and 50% for kMutation are best. Under these conditions the performance of the adaptation approach on the first problem is disappointing: the average generation at which a solution was found was 59. This is almost twice the value for the static environment case. This adaptation test used a morphing constant of 50, which means that by generation 20 the polygons would have reached their final form. Furthermore, this implies that approximately forty generations were used with the static obstacles to find a solution, so that the first 20 generations actually hurt the performance of the standard kMutation algorithm!

There are several techniques which improve the performance of the adaptive algorithm. First, preventing the smoothing portion of the kMutation operator helps avoid collapsing the space of paths down to the straight line path at the beginning of mutation. However, smoothing is still desirable once the polygons have once again become static. As a result, the adaptation algorithm prevents smoothing until the polygons have reached their full extent, after which smoothing is applied. This only makes the adaptation algorithm slightly more effective: it finds the solution, on average, at generation 53.08 with a 50% chance of kMutation and a 0% chance of crossover, which was the combination which provided the best results. (Letting crossover reach 10% slowed the search down to about 60.) Also, allowing no smoothing makes the algorithm much slower, but it does make the crossover operation useful. The optimal probabilities were 20% crossover and 30% kMutation, with an average generation of 152.3. Further experiments will use the smoothing once the obstacles have reached their full extent.

Figure 4 shows the performance of the same algorithm as above, except that the kMutation operator only changes one line segment instead of all of them. The crossover probability is the value on the x-axis divided by 100. So 500 is 5%. The kMutation probability is 50% minus that value. Each data point represents 100 trials. It is surprising that the algorithm performs better! This results in a great reduction in computational cost, since the full kMutation operator is very costly. Furthermore, the average generation of success of 47 is approaching the same performance as the static algorithm with the more costly kMutation operator. Subsequent experiments will use the simplified kMutation operator.

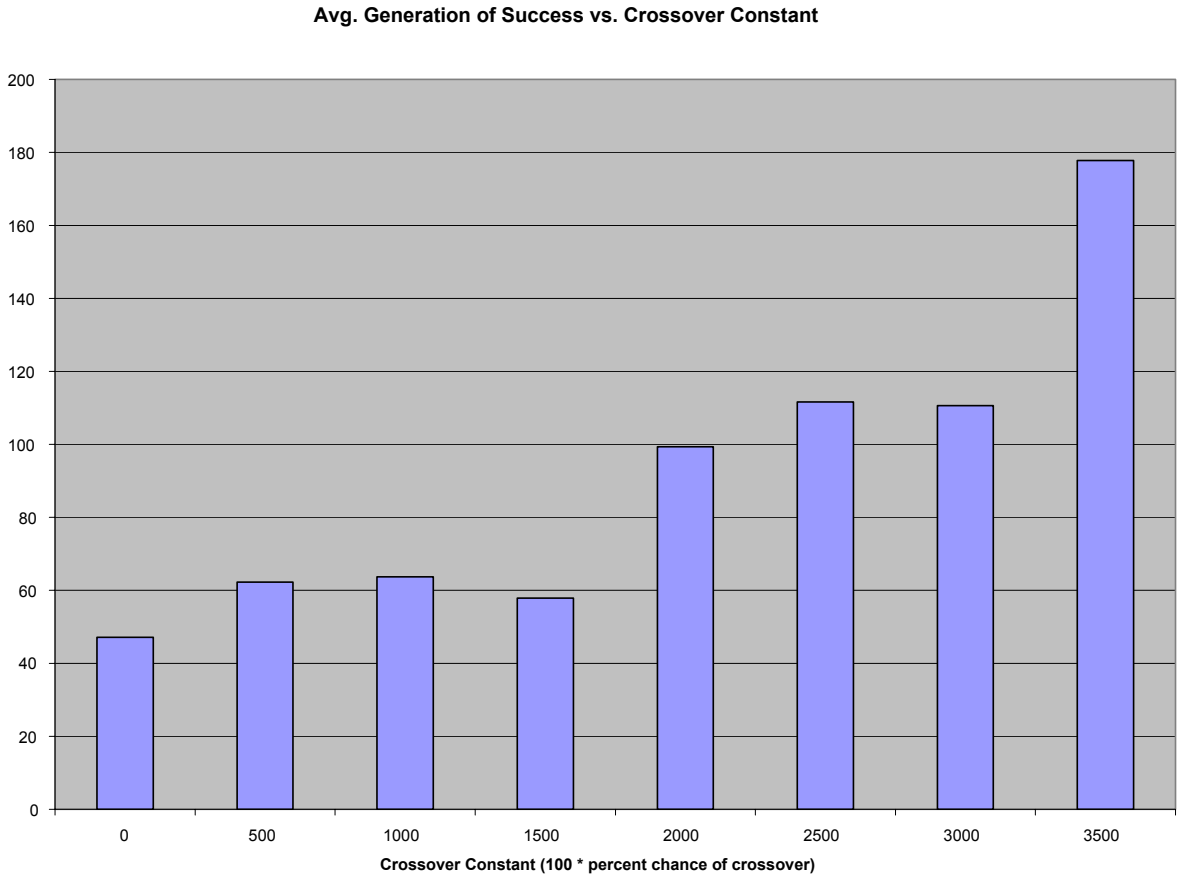

Figure 4

For comparison, the simplified kMutation search without adaptation requires an average of 124.6 generations to find a solution. This suggests that the adaptive approach is already more efficient than the static approach.

A factor in the efficiency of the adaptation approach which we have not yet discussed is the rate of morphing. Figure 5 shows the performance of the adaptive algorithm on the third and hardest map with smoothing after the morphing of obstacles is complete for varying morphing constants.

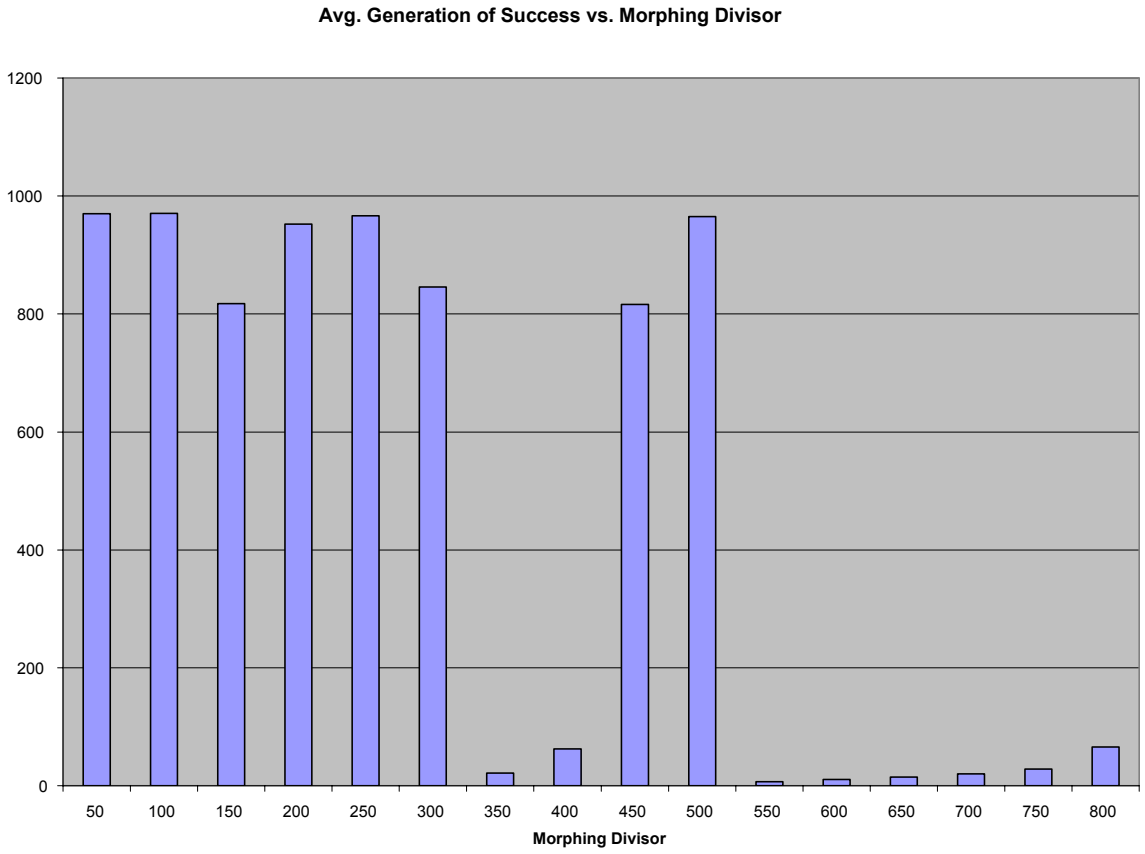

Figure 5

The results show that the algorithm performs best with a morphing divisor of 550. To compute the actual morphing rate of the polygons with respect to the number of generations, we divide 1000, the maximum number of generations allowed, by the morphing divisor. Then, at generation 0, the polygons are points. By generation 1, they are somewhat under half extended. By generation 1, they are nearly fully extended. Finally, by generation 3, they are fully extended and smoothing begins to be used. Since the average number of generations was 6.78 for that data point, approximately half of the search time was spent while the polygons were expanding, and approximately half was spent with the polygons in a static state. The average number of generations required to find a solution to the third map for the static algorithm using the simple kMutation operator was 918.19. Therefore, the speed of the algorithm increased by over two orders of magnitude with an optimal morphing rate. The average generation of success was similarly improved for the first two problems, save that it was usually only by one order of magnitude. This suggests that the benefits derived from using an adaptive technique increase with the difficulty of the problem. These experiments suggested a morphing constant of eleven, as well. Perhaps the most interesting feature of Figure 5 is that the morphing constant produces very good results with any value between 7 and 14 EXCEPT for values 9 and 10. The phase transitions are apparently extreme. Since each data point represents 500 trials, this effect is unlikely to be due to chance. More investigation is needed to determine the reasons for this anomaly.

Conclusion

This paper has demonstrated the effectiveness of using of a changing environment designed to make the pathfinding problem increasingly difficult with each successive generation. This allows the population to adapt to the changing environment gradually, which effectively focuses the genetic search and greatly increases its speed.

Developing heuristics to choose appropriate values for the genetic search algorithm's parameters may be the hardest problem expanding the usefulness of this pathfinding algorithm to arbitrary maps. While this paper is only a tentative first step in researching the applications of this potentially widely applicable strategy, the preliminary results are very promising.

Future Research

The most important future potential is determining a good heuristic for the morphing constant's initial value, perhaps based on the number and clustering of obstacles, and experimenting with functions other than linear interpolation in the hopes of achieving better results.

Perhaps the second most important potential for future research involves modifications to the simplified kMutation operator. It uses no information about the relative location of the obstacle which the line segment intersects. If the operator were extended so that it could compute a line through the midpoint of the intersections and the centroid of the obstacle, a new point could be chosen along that line beyond the midpoint. That point would be much more likely to make the new path avoid the obstacle than a point generated from a uniform distribution at sufficiently low cost that it should improve the performance of the algorithm. Most importantly, it ensures that a path is "bumped" in the correct direction by an expanding polygon, which could immensely improve the performance of the adaptation algorithm.

The knowledge based mutation operator could also be extended to use information about the portion of the line segment inside of an obstacle, perhaps as computed when determining the fitness, to avoid the obstacle. If only a part of the path is in the obstacle, while its endpoints are not, a new point should be added to avoid the obstacle. If an endpoint is inside an obstacle, the endpoint must be moved. The length of the line segment inside of the obstacle could also be used as a heuristic measure of how far any mutated endpoint should be allowed to range from its original position, or of how large the area of the uniform distribution should be for a new point which is inserted into the path.

An operator which may be useful when combined with the changing obstacles would use the previous intersections of a line segment combined with information about the new intersections to determine how the length of the line segment inside of obstacles had changed once the polygons had changed size. This could allow the mutation operation to determine a probability distribution for the location of a new point that would maximize the chances of avoiding the obstacle. That is, knowing the rate of change of the length of the line segment in the obstacle combined with the position of that length provides information to guess at the direction of growth of the polygon and its rate of growth. Choosing a point along that line of growth (perhaps in both directions) at a distance determined heuristically by the rate of growth could prove useful.

Determining the size of the population appropriate for an arbitrary search space is impossible. However, the size of the population could be optimized as the algorithm runs. If the number of paths falling into certain schema exceed a maximum value, the extra paths could be deleted and the size of the population decreased. If the number of paths is too low, new paths could be added and the population size increased. This requires applying a concept of schema to the algorithm. One simple approach would involve discretizing the space into a rough grid. Then each path fits a schema consisting of a finite number of integer pairs describing a grid coordinate, allowing similar paths to be identified. This, however, would be computationally expensive.

Another possibility would generate a set of initial population members deterministically. Supposing that the centers of each obstacle are known, perhaps choosing to create some number of paths for each possible combination of passing above or below the polygons (ordered left to right) would produce good results. Of course, for polygons as opposed to arbitrary obstacles, these paths could be constricted around the polygons very rapidly and no genetic search would be necessary.

The crossover operation, at relatively high computational expense, could be extended to choose only sites at which two paths are relatively close to one another. By performing the crossover on that point, the resultant paths would be more continuous, which would make them less likely to strike obstacles along a potentially long line segment joining the two pieces. This may make the crossover operation once again useful in the adaptive algorithm.

Application of restricted mating can also improve the performance of the algorithm by encouraging the pursuit of multiple possible paths to the goal.

References

Goldberg, David E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Boston, MA: Addison-Wesley Longman, Inc.

Koza, John R. 2002. *Genetic Algorithms and Genetic Programming*. Stanford, CA: Leland Stanford Jr. University Press.

Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: The MIT Press.