An Empirical Study of Crossover and Mass Extinction in a Genetic Algorithm for Pathfinding in a Continuous Environment

H. David Mathias Vincent R. Ragusa
Department of Mathematics & Computer Science
Florida Southern College
Lakeland, Florida 33801
Email: hmathias@flsouthern.edu
Email: vragusa@mocs.flsouthern.edu

***Abstract*—Genetic algorithms are an often used tool for the problem of pathfinding. Their ability to find good solutions to multiobjective optimization problems makes them well suited to this task. Of course, genetic algorithms embody a broad range of techniques and strategies including crossover, mutation and mass extinction, each with multiple parameters and implementations. In this paper, we examine the effects of crossover and mass extinction on a genetic algorithm for planning a path through known obstacles in an unconstrained, continuous, static environment. Using a mutation-only genetic algorithm as a baseline, we study the effect of crossover with and without mass extinction events that occur at several different probabilities. We find that while both offer sometimes significant improvement in some cases, neither is universally beneficial.**

1. Introduction

Pathfinding is a problem with a long history. From the Bridges of Königsberg to the Traveling Salesman to network routing, many variations, with different goals and constraints, have been extensively studied and many techniques applied. In recent years, robotics has brought interest from researchers in a new and growing field.

In robotic pathfinding, a robot must plan a path from a starting location to an ending location in the presence of obstacles. The primary objective is to optimize some quantity, most often path length. The environment is typically bounded and discrete. In many instances of the problem, there are multiple objectives. For example, in addition to path length, one may wish to minimize the number of obstacles hit, the number of waypoints or the sum of the angles of the turns the robot is required to make. Not surprisingly, attempting to optimize multiple objectives significantly complicates the search, particularly when objectives conflict. As a simple example, consider a path that consists of a straight line from source to target that hits numerous obstacles. Decreasing the number of obstacles hit necessarily increases path length.

Thus, one of the difficulties introduced by the use of multiple objectives is that of evaluating candidate solutions. With a single objective, this task is trivial. To compare two candidates, simply compare the values of the objective for each, a method that does not scale to the multiobjective environment. The most obvious solution is to use a utility function to scalarize the multiple values. The utility function weights each objective value via a coefficient (and less commonly, an exponent other than 1) to obtain a single, easily compared result. While this technique makes comparisons rather straightforward, it introduces another significant complication: how to properly weight the objectives to best direct the search.

Many strategies have been used to generate candidate solutions for pathfinding problems, from simple search strategies to various artificial intelligence methods. In this work, we focus on genetic algorithms [2] [6] [9] [17] [22]. Konak, Coit and Smith [11] catalog a number of genetic strategies for multiobjective optimization [5] [10] [18] [3] [23] [24] [20]. More recent results have built on this foundation [21] [4] [7]. In this work, use the NSGA-II algoritm by Deb *et al* [3] as a starting point. There are more recent results, however, NSGA-II is efficient and has been widely implemented. That the algorithm is well-tested is important because in our system, it runs onboard a physical micro aerial vehicle that flies the paths found. That it is efficient is significant due to the limited power of the single board computer onboard the vehicle.

Many of the referenced algorithms, including NSGA-II, leverage the work of economist Vilfredo Pareto. While *Pareto efficiency*, as originally formulated, dealt with allocation of multiple resources in a society, it has been adapted for use in engineering and computer science.

Let O be the set of optimization objectives. We say that one candidate solution X *dominates* another candidate solution Y if $(\forall o \in O,\ X[o] \leq Y[o]) \wedge (\exists o\ s.t.\ X[o] < Y[o])$ [15]. A solution that cannot be dominated is known as *Pareto optimal*. The set of all Pareto optimal solutions defines the *Pareto front*. The idea is to then assign each candidate solution to one of some number of *domination fronts* in which each candidate in front i is dominated by at least one candidate in front $i-1$. Thus, the fronts impose a partial order on the candidates. Front 0 contains all non-dominated candidates. In a search for an optimal solution, front 0 constitutes the best approximation of the Pareto front.

A *genetic algorithm* operates much like Darwinian evolution. A population of candidate solutions, initially random, evolves through some combination of selection, mating and mutation operators. Each member of the population is encoded in a chromosome. The information stored in the chromosome – the genes – differs greatly with the problem being solved. *Crossover*, in which genes from each of two parents are combined to produce a child, simulates mating. Random mutations alter the chromosome of an existing member of the population. Each generation of the population creates a new generation via some combination of crossover and mutation. Most algorithms maintain a fixed population size, n. This is achieved via selection, a form of survival of the fittest.

While crossover is commonly implemented in genetic algorithms for pathfinding, it does not appear that a study of it's effectiveness has been undertaken. Tonupunuri [19] compared a selection-only algorithm to variants that incorporated first crossover and then mutation. While crossover was shown to improve the result of pathfinding in a mobile sensor network, it was compared to a selection only result, which is to say a non-genetic algorithm. There was no examination of the effect of adding crossover to an algorithm with other genetic components, namely mutation.

Biological evolution is, of course, far more complex than mating and mutation. Other events, such as mass extinctions, influence the evolutionary thread of a species. In nature, mass extinction occurs due to phenomena such as disease, changing habitat and destructive events such as meteors and volcanoes and may result in elimination of all or part of a species. If a species survives, it may benefit subsequently from accelerated evolution [12]. This naturally suggests investigating use of mass extinction events in genetic algorithms [8]. Algorithmically, these events can occur at random, at some fixed interval, or as triggered by some metric, and typically kill some fraction, but not all, of the population. Regeneration may occur through reproduction, random generation of new members, mutation of surviving members or some combination of these methods. The idea is that the large-scale infusion of new genetic material may help generate strong candidate solutions. A potential negative effect is that strong members of the population may be eliminated.

In this paper, we examine the effects of crossover and mass extinction on a genetic algorithm for pathfinding in a very general, continuous environment. The broad use of crossover in such algorithms begs the question of its efficacy for pathfinding. For mass extinction, it appears that the question is even less understood. We are not aware of any results that incorporate mass extinction for this problem and its utility is, therefore, unknown. We examine these questions through a series of experiments.

2. A Genetic Algorithm for Pathfinding

Genetic algorithms encompass a range of techniques with an even broader range of parameters that influence the results. Here we present our pathfinding problem and the details of our genetic algorithm.

An instance of the pathfinding problem we consider consists of a starting location, s; a destination location, d; a (possibly empty) list of obstacles, *obs*; and a (possibly empty) list of intermediate targets, *goals* in a largely unconstrained continuous environment. s and d are represented by GPS coordinates and can exist anywhere in the environment. Obstacles and intermediate goals can require flight in any direction and co-location of the s and d can require a circuitous path. Obstacles are circular or polygonal. If obstacle o_i is circular, it is represented by the GPS coordinates of it's center and a radius in meters. If polygonal, it is represented by a list of vertices, each of which is a GPS location. Intermediate targets are locations to which the vehicle must fly, prior to the destination, to successfully complete it's mission. These may represent locations near RF-enabled sensors or from which a photograph must be taken, for example. Each element of *goals* is a circle represented in the same format as a circular obstacle.

A solution to the problem consists of a sequence, $S = \langle w_0, \ldots, w_n \rangle$ of waypoints, where $w_0 = s$, $w_n = d$. Each waypoint w_i is represented as a real-valued pair, (lat, lon), consisting of the latitude and longitude of the location. In general, waypoints are points of articulation in the path, though it is possible for a waypoint to bisect a straight path segment. A solution is considered *valid* if the path defined by S avoids all obstacles in *obs* and reaches all targets in *goals*.

In many genetic algorithms, the chromosome is treated as a bit string and genetic operators are applied at the bit level. For pathfinding, numerous representations have been evaluated. Hermanu *et al* [6], Sedighi *et al* [16], and Ahmed and Deb [1] have reported on the effectiveness of different chromosomes. Directly applying these results to our problem is difficult due to the discrete environments they consider. Zheng *et al* [22] work with a continuous environment. Their chromosome consists of a sequence of waypoints, each with a state variable indicating feasibility of the incident path segments. Similarly, our chromosome consists of waypoint sequence S, in which each waypoint is represented as described above. We do not encode a state variable. Our chromosome is of variable length reflecting that different paths will likely require different numbers of turns and, therefore, waypoints.

In our approach, operators are applied to waypoints, as opposed to bits, as waypoints are the atoms in our representation. The reason for this is to constrain the results produced by the operators. For example, flipping a single high-order bit in the chromosome could result in a huge change in the position of a waypoint. Because our algorithm is designed to be used for navigation on an actual vehicle with very limited flight time, such large-scale changes are impractical. No waypoint is viable if it requires movement to a position beyond what is possible within the limits of the battery life of the vehicle.

The objectives we seek to optimize are: path length, number of obstacles hit, number of targets hit and smoothness [9]. Smoothness, defined as $\sum_{i=0}^{n-1} \angle w_i w_{i+1}$, is the sum of the angles between pairs of consecutive waypoints. We wish to minimize this quantity since sharp turns require more time and, possibly, more maneuvering for certain types of vehicles. So that we can minimize all objectives, we negate the number of targets hit.

Our initial population, generation 0, consists of randomly generated members. The chromosome for each member contains between 1 and 5 randomly generated waypoints. Each of these waypoints must be within some threshold distance of starting point s. Using the seminal work of Deb *et al*, NSGA-II [3], as a starting point, our basic algorithm, which serves as a baseline in our experiments, applies only mutation and selection operators to maintain a population of candidates. The population is sorted into domination fronts as described in Section 1.

From generation g_j, we create generation g_{j+1} as follows. Let generation j be the parent population, p_p. We first create a child population p_c such that $|p_c| = |p_p| = n$. We then merge p_p and p_c into a combined population p with size $2n$. p is then resorted into domination fronts. Within a front, a niching measure [14] known as *crowding distance* [3] can be used to impose a partial order on the members. The crowding distance of a member is a measure of the dissimilarity of that member to other members of the population. Greater values are preferred to help ensure diversity in the gene pool. Let k be the largest index such that $\sum_{i=0}^{k} |f_i| = y \leq n$ where each f_i is a domination front. Then g_{j+1} consists of all members from f_0 through f_k and $n - y$ members from f_{k+1} chosen by crowding distances in decreasing order.

Child creation occurs via mutation of a member of p_p. To select a member to mutate, we use a tournament with *tournament pressure* equal 4. That is, we choose four members of p_p and find the "best" via a series of comparisons. Let member m_a be in front f_a and member m_b be in front f_b. We say that m_a is better than m_b if $a < b$. If $a = b$, we use largest crowding distance to choose the winner. When the best member of the 4 has been chosen, it is copied and the copy is mutated. Both members are retained until the next application of the selection operator at which point their fitness will determine survival.

We implement four mutation operators: *add, delete, swap,* and *move*. When a member is chosen for mutation, exactly one of the operators is applied. The probabilities of the operators are 0.1, 0.05, 0.1 and 0.75, respectively. While *add* inserts a new waypoint into S, the other operators each alter an existing waypoint. Distinguished waypoints w_0 and w_n cannot be mutated.

The *delete*, *swap*, and *add* operators are easily explained. *delete* removes randomly selected w_i from S for some $0 < i < n$. *swap* randomly selects a pair of adjacent waypoints w_i and w_{i+1}, $0 < i < i+1 < n$, and exchanges their positions in S. The *add* operator works by randomly selecting a pair of adjacent waypoints w_i and w_{i+1}, $0 \leq i < n$. It then creates a new waypoint, w_k, at the midpoint of the path segment between them. The *move* operator is then applied to w_k.

The *move* operator is somewhat more complex. A waypoint, w_i, $0 < i < n$, is chosen at random. The latitude and longitude values for w_i are changed independently according to a Gaussian distribution wtih mean 0 and standard deviation σ. The value for σ is selected by fair coin flip from among two candidates, *small_move* and *big_move*. *small_move* is fixed at 0.00002 degrees, approximately 2 meters. The value of *big_move* depends on the problem instance. For instances with small obstacles, it is 0.0002 degrees ($\sim$ 20 m) and for instances with large obstacles, it is 0.00067 degrees ($\sim$ 69 m).

The algorithm, as described above, serves to provide a baseline for comparison when evaluating the effects of crossover and mass extinction on pathfinding.

2.1. Crossover

Crossover is the genetic algorithm corollary to sexual reproduction. In general, it involves dividing the chromosomes of two members of the population into multiple parts and exchanging them to create child members. One complication in pathfinding is that chromosomes may not have a fixed length. Certainly this is the case in our implementation. This variability adds a degree of genetic diversity to the population.

In the algorithm described here, we use single-point crossover. Due to the structure of the chromosome we use, uniform crossover is not possible. Further, due to the short length of chromosomes typical in the problem instances we solve, two-point and multipoint crossover are impractical. However, thanks to the diversity introduced by variability of chromosome length, we do not believe that the use of single-point crossover is limiting.

Let p_k be a member of the population represented by its chromosome, waypoint sequence $\langle w_0, \ldots, w_{n_k} \rangle$. We define cutpoint t_i as a partition of p_k into $p_k^1 = \langle w_0, \ldots, w_{i-1} \rangle$ and $p_k^2 = \langle w_i, \ldots, w_{n_k} \rangle$. Neither subsequence can be empty. With this, given members p_1 and p_2, our algorithm chooses random cutpoint t_i in p_1 and t_j in p_2 and creates two children c_1 and c_2 such that $c_1 = p_1^1 + p_2^2$ and $c_2 = p_2^1 + p_1^2$. It is probable that $|c_1| \neq |c_2|$ but this is inconsequential. After creation, the children may, with some probability, undergo mutation.

As in the mutation-only version of the algorithm, we create a child population of size n, combine it with the parent population of size n to create, temporarily, a population of size $2n$. The algorithm then resorts the combined population into domination fronts, and chooses the top n members to populate the next generation.

2.2. Mass Extinction

Though the concept of mass extinction exists in the genetic algorithms literature [8] [12], it has not been widely adopted in general and not at all for pathfinding. Therefore, the viability, or lack thereof, for mass extinction should be investigated for this problem. The intended goal of extinction events is to jump start evolution when the process

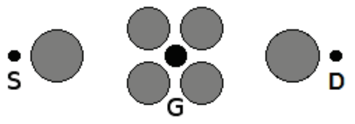


Figure 1. Input 1 with starting point S, destination D, single goal G and six circular obstacles (in gray).

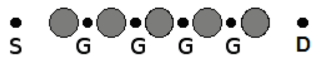


Figure 2. Input 2 with goals to produce a slalom.

stalls, to get out of evolutionary dead-ends and avoid local minima. We implemented, and ran experiments with, three versions of mass extinction. The first is entirely random while the second and third use a form of elitism to prevent eliminating the best genetic members of the population. In all three cases, the operator is invoked with probability p at an interval of 50 generations.

2.2.1. Random Regeneration. Our first implementation of mass extinction is the simplest. 80% of the population is eliminated entirely at random. All eliminated members are replaced with randomly generated new members. While perhaps providing a reasonable baseline for comparison with other extinction implementations, this version may be problematic for vehicle flight. Completely random extinction events may eliminate all valid solutions within the population. Such an occurrence late in a run of the system would leave the vehicle without a valid path to fly. To avoid this, onboard the MAV we use only those extinction implementations that incorporate elitism.

2.2.2. Regeneration via Crossover. In our second implementation of extinction, each extinction event eliminates 80% of the population. In a form of elitism, the best 3 members of the population are exempted from extinction. Repopulation is a two-step process. First, 60% of the deleted members are replaced by randomly generated new members. The remaining 40% are created via crossover. Candidates for crossover are those members that survived extinction together with the newly created random members.

2.2.3. Regeneration via Mutation. Elitism plays a greater role in our third implementation. In this version, each extinction event eliminates all but the best 10 members of the population. Once again, repopulation is a two-step

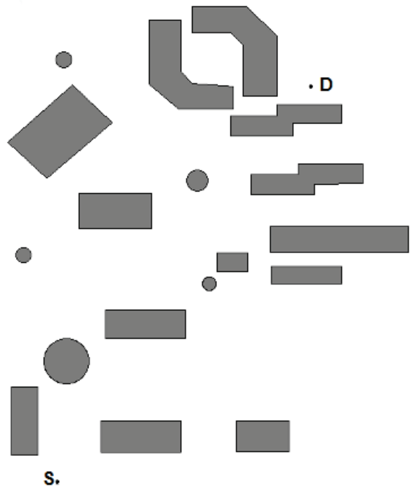


Figure 3. Input 3 – a representation of part of the St. Leo University campus.

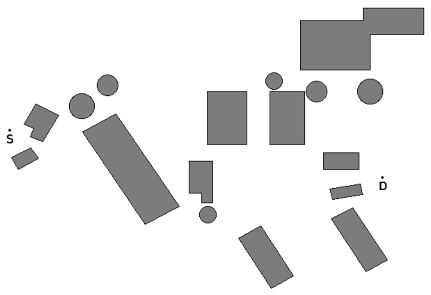


Figure 4. Input 4 – a representation of part of the Florida Southern College campus.

process. First, 10 randomly generated members are added to the population to ensure some degree of genetic diversity. Then, the balance of the population is created by randomly selecting one of the 10 members preserved through elitism, cloning it and mutating the clone.

3. Experimental Design

Our goal is to experimentally determine the effects of crossover and mass extinction when used in a genetic algorithm for pathfinding. To do this, we ran the algorithms on a set of five sample inputs (see Figures 1 – 5). Each trial consisted of running the algorithm ten times for each input: with extinction probability p equal to $0.0, 0.05, 0.10, 0.15, 0.20$, each with and without crossover. Recall that extinction occurs with probability p every 50 generations. For these tests, the population size was fixed at 100.

For each mass extinction implementation, we ran 40 trials for a total of 400 runs for each of the 5 inputs. In each trial, all runs for a given input used the same random seed.

For each input, we established a *best found solution* via a large number of runs of our mutation-only algorithm. We

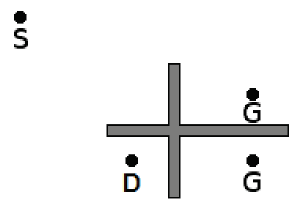


Figure 5. Input 5 with GPS coordinates mapped to the field on which we typically perform flight tests.

use the path length values from these solutions to establish two of the metrics used for experimental evaluation. In each trial, we track the mean number of generations for three measures: a valid solution, denoted *mng_valid*; a solution with path length within 10% of the best found solution, *mng_10*; and a solution with path length within 5% of the best found solution, *mng_5*. Each run of the algorithm is limited to 3000 generations but will terminate prior to that if a 5% threshold solution is found.

4. Results

Our results show that both extinction and crossover offer some benefit for pathfinding in some cases. Unfortunately, there are also instances in which these operators have a negative effect on the number of generations required, sometimes dramatically so. Crossover is predominately negative, though there are notable exceptions. For extinction, in most cases, the effect ranges from slightly negative to very positive.

The data presented are trimmed means. From our initial sample of 40 runs for each experiment, we removed the top 5% and bottom 5%. Graphs showing *mng_5* data for each input appear in Figure 6. Due to space considerations, we cannot present graphs for all experiments. Therefore, graphs are all for *mng_5* to allow comparison across inputs. For each mean, we calculated a confidence interval. To isolate the effect of extinction, we have compared the mean values with extinction probability $p = 0$ to a mean calculated for all probabilities $p \in \{0.05, 0.10, 0.15, 0.20\}$ (see Figures **??** **??**). We also performed two sample T-tests to explore the impact of crossover (see Table 1).

Crossover improved mean number of generations for measures *mng_10* and *mng_5* for input 1 (Figure 6) independent of extinction implementation. For input 2, crossover was beneficial for *mng_valid*, with extinction implementations 1 and 2, but detrimental for both *mng_10* and *mng_5*. For inputs 3 and 5, crossover shows a benefit for *mng_valid* for all extinction probabilities and all implementations. However, it is detrimental with respect to *mng_10* and *mng_5* thresholds; dramtically so for input 5. For input 4, crossover is extremely negative with respect to *mng_5*.

In the presence of crossover, extinction implementation 1 is, in general, the worst of the three versions, though in several cases it is on par with the others and for input 3 (Figure 6) it is arguably the best implementation. On the other hand, it is the only implementation that exhibits very large negative effects, for example *mng_valid* for inputs 2 and 4. The overall inferiority is not surprising given the absence of elitism. Comparing extinction 2 and extinction 3, we see that extinction 2 is, with few exceptions, more effective, though the differences are small, often negligible. In the absence of crossover, there is little difference between extinction versions for inputs 2, 4, 5. For input 4, all extinction versions are beneficial.

In some cases, the impact due to extinction appears to improve with number of generations. For this reason, *mng_5* is the measure most positively impacted and *mng_valid* least. Input 4 offes the best illustration. All extinction versions have a positive impact but the effect is strongest in the presence of crossover. The runs with crossover for that input take significantly more generations. For input 3, all extinction versions show some promise, though at varying extinction probabilities, with extinction 3 showing the greatest improvement at probability 0.1. Extinction 3 also shows the best improvement for input 5, though in this regard it benefits from an inferior baseline.

Figure 7 shows the effect of extinction for inputs 1 and 4. In each pair, the left bar is the trimmed mean for extinction probability 0 and the right bar is the trimmed mean for all other probabilities studied. 95% confidence intervals are also displayed. Again, we see that for input 4, extinction is positive in all cases, while for input 1 there is no trend.

Table 1 contains data for two sample T-tests for inputs 1 and 4. In each case, the two samples are for the same parameters except for crossover. In one sample crossover is used and in the other it is not. Instances for which the difference is statistically significant for a 95% confidence interval are in bold. For *mng_10* and *mng_5*, crossover makes a statistically significant difference in almost all cases, beneficial for input 1 and detrimental for input 4. This trend holds for all other inputs.

5. Conclusion

In this work, we have attempted to evaluate the effects of crossover and mass extinction on a genetic algorithm for pathfinding in a continuous environment. Due to the large number of possible algorithms and parameters as well as variations in environments, we cannot make a definitive statement regarding the utility of these operators. Broader experimentation is required to explore these variables. However, our results show that crossover may not be desirable for some instances of pathfinding while mass extinction shows some promise.

The inputs we used for testing are varied. Numbers 1 and 2 are linear and symmetric yet the results for these inputs are very different. Crossover helps with 1 but not 2 and extinction is slightly negative for 1 and neutral for 2. Inputs 3 and 4 are drawn from the real world and are

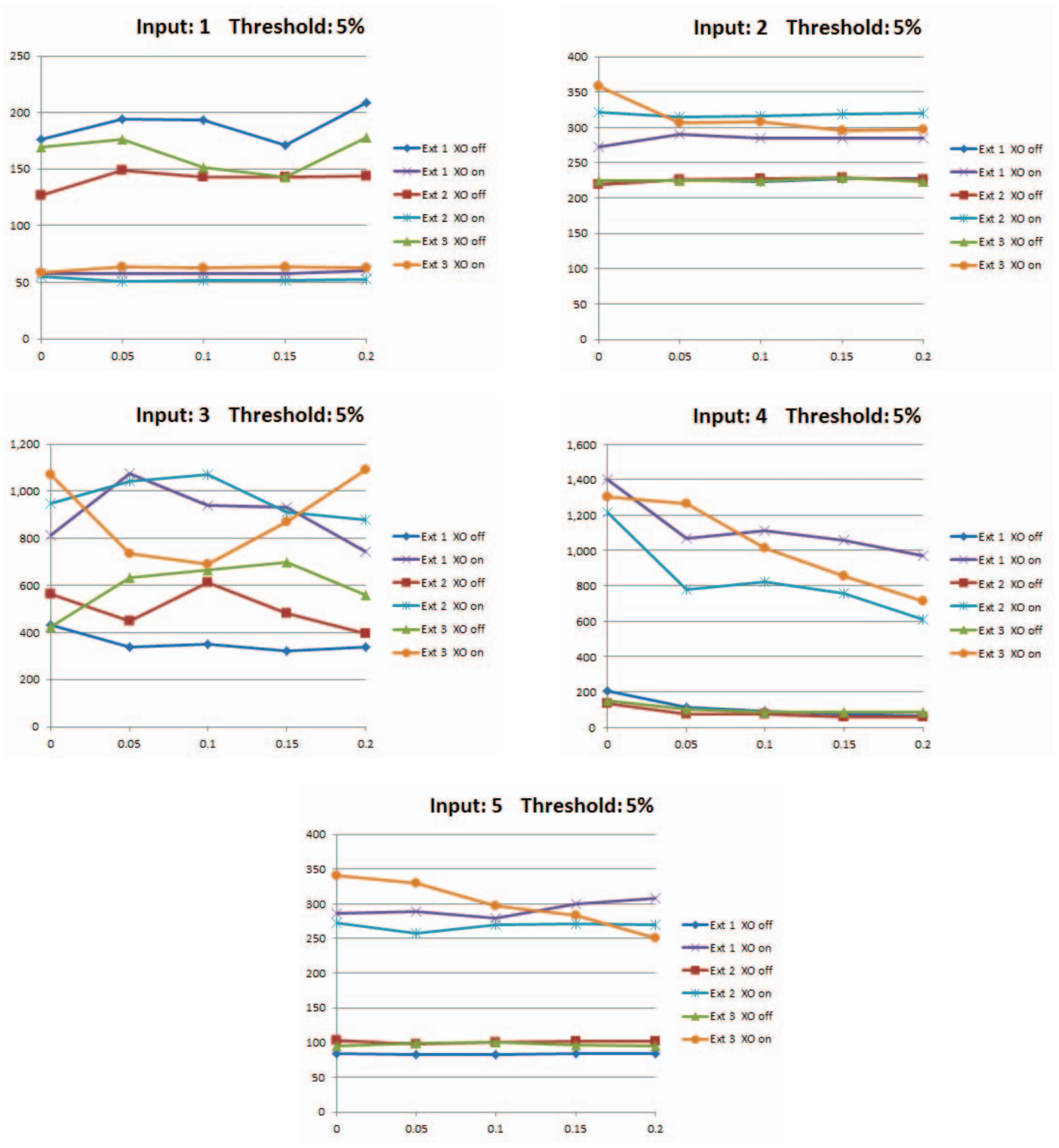
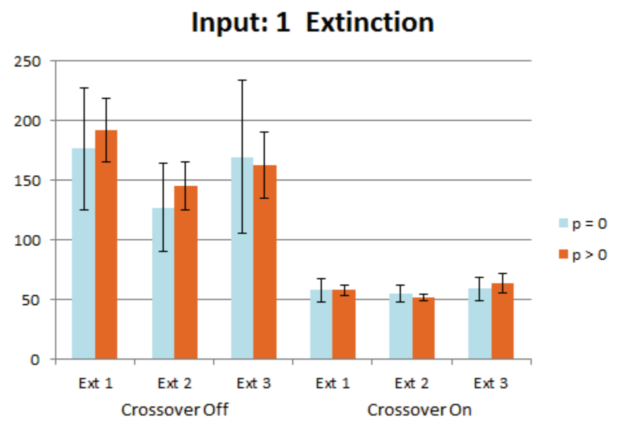


Figure 6. *mng_5* graphs for all five inputs. Each graph shows all three extinction implementations with crossover on and crossover off. Note that the scales differ.

significantly more complex. In both cases, crossover is a negative influence. Input 5 is simple but not symmetric. In this case, crossover is negative while extinction is neutral.

When extinction exhibits a benefit, it is often best for *mng_5*, the threshold that requires the largest number of generations to reach. This makes intuitive sense: because extinction occurs infrequently, longer times provide greater opportunity for extinction to benefit the population. In fact, for *mng_5*, benefit due to extinction increases slightly with extinction probability. The reverse is true for *mng_valid*.

In our experiments, crossover and extinction coexist reasonably well. For some inputs, extinction improves re-



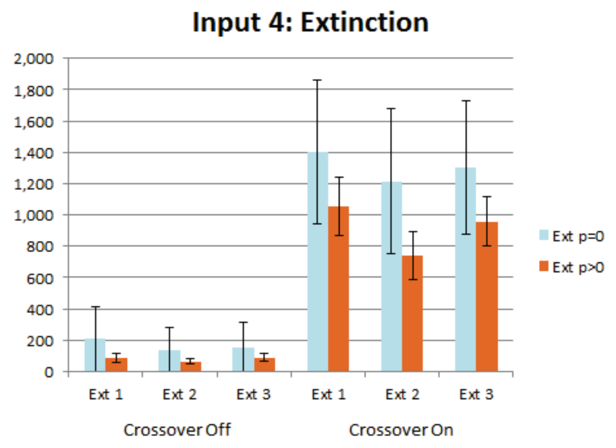


Figure 7. The effect of extinction for inputs 1 and 4. For these graphs, we treat extinction as on or off, with data for all probabilites greater than 0 aggregated into a single mean.

sults when used with crossover while in others extinction is neutral. In no case is the combination significantly negative for measure *mng_5*, though for most inputs crossover is negative. However, this effect is independent of extinction.

There is a great deal of future work to continue this research. As we mention above, greater variation in extinction implementations and parameters, such as probability and the frequency with which extinction is invoked, could expose trends that we did not see. For example, rather than invoking extinction at regular intervals, we could determine when to use it based on a decreasing rate of change in the population. Alternatively, the increasing utility of extinction with number of generations might suggest increasing the extinction probability with time.

Another area for future work is the extent to which random regeneration aids extinction. Extinction implementations 2 and 3 vary significantly in the amount of random regeneration. We hypothesize that the new genetic material introduced by random members could benefit evolution. Experiments in which both versions are used with varying numbers of random new members might identify a trend and the ideal random percentage.

Acknowledgments

The authors would like to thank Florida Southern College for financial and other support for this research. They also thank Annie Wu for valuable feedback and Isabel Loyd and Susan Serrano for assistance with the statistical analysis.

References

[1] F. Ahmed and K. Deb, Multi-objective optimal path planning using elitist non-dominated sorting genetic algorithms. Technical Report 2011013, Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology Kanpur, 2011.

[2] H. Burchardt and R. Salomon, Implementation of path planning using genetic algorithms on mobile robots. in 2006 IEEE Congress on Evolutionary Computing, 1831-1836.

[3] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6(2), 182-197, 2002.

[4] K. Deb and H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based non-dominated sorting approach, Part I: Solving problems with box constraints. IEEE Transactions on Evolutionary Computing 18(4), 577-601, 2014.

[5] C. Fonseca and P. Fleming, Multiobjective genetic algorithms. in IEE colloquium on Genetic Algorithms for Control Systems Engineering, Digest No. 1993/130, London, UK, 1993.

[6] A. Hermanu, T. Manikas, K. Ashenayi, and R. Wainwright, Autonomous robot navigation using a genetic algorithm with an efficient genotype structure. Intelligent Engineering Systems Through Artificial Neural Networks: Smart Engineering Systems Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Complex Systems and Artificial Life. ASME Press, 2004.

[7] H. Jain and K. Deb, An evolutionary many-objective optimization algorithm using reference-point-based non-dominated sorting approach, Part II: Handling constraints and extending to an adaptive approach. IEEE Transactions on Evolutionary Computing 18(4), 602-622, 2014.

[8] B. Jaworski, L. Kuczkowski, R. Smierzchalski, and P. Kolendo, Extinction event concepts for the evolutionary algorithms. Przeglad Elektrotechniczny (Electrical Review), 88(10b), 2012.

[9] H. Jun and Z. Qingbao, Multi-objective mobile robot path planning based on improved genetic algorithm. in 2010 IEEE International Conference on Intelligent Computation Technology and Automation, 752-756.

[10] J. Knowles and D. Corne, The pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimization. in 1999 Congress on Evolutionary Computation.

[11] A. Konak, D. Coit and A. Smith, Multi-objective optimization using genetic algorithms: A tutorial. Reliability Engineering & System Safety 91, 992-1007, 2006.

[12] J. Lehman and R. Miikkulainen, Extinction events can accelerate evolution. PLoS ONE 10(8), August 2015.

[13] D. Mathias and V. Ragusa, Micro aerial vehicle pathfinding and flight using a multi-objective genetic algorithm. Under review, 2016.

Input 1													
		Valid				10%				5%			
Ext Ver	Prob	Est Diff	P-value	T-value	Deg F	Est Diff	P-value	T-value	Deg F	Est Diff	P-value	T-value	Deg F
1	0.00	5.0	0.121	1.57	55	88.1	**0.000**	4.85	37	118.1	**0.000**	4.65	37
	0.05	22.1	0.090	1.74	36	98.7	**0.000**	5.93	37	136.9	**0.000**	4.74	36
	0.10	40.8	0.051	2.02	35	106.4	**0.000**	5.69	36	135.7	**0.000**	4.70	36
	0.15	20.4	**0.029**	2.27	37	92.7	**0.000**	6.49	38	113.6	**0.000**	5.67	38
	0.20	43.8	**0.024**	2.36	35	108.3	**0.000**	5.57	37	147.8	**0.000**	4.98	36
2	0.00	1.7	0.599	0.53	55	55.5	**0.000**	4.60	39	72.0	**0.000**	3.87	37
	0.05	2.1	0.538	0.62	53	65.3	**0.000**	5.37	37	97.5	**0.000**	4.52	36
	0.10	2.1	0.542	0.61	53	62.9	**0.000**	5.31	38	91.4	**0.000**	4.70	36
	0.15	2.1	0.542	0.61	53	63.2	**0.000**	5.34	38	91.6	**0.000**	4.71	36
	0.20	2.1	0.542	0.61	53	61.2	**0.000**	5.28	38	91.8	**0.000**	4.34	36
3	0.00	2.1	0.518	0.65	58	74.1	**0.000**	4.08	38	110.4	**0.001**	3.48	36
	0.05	2.6	0.461	0.74	55	74.0	**0.000**	5.08	38	112.8	**0.001**	3.51	40
	0.10	2.6	0.466	0.73	55	68.7	**0.000**	5.42	39	88.9	**0.001**	3.70	44
	0.15	2.0	0.538	0.62	59	65.9	**0.000**	5.48	40	79.8	**0.001**	3.59	46
	0.20	2.0	0.538	0.62	59	69.5	**0.000**	5.37	39	114.6	**0.003**	3.16	38

Input 4													
		Valid				10%				5%			
Ext Ver	Prob	Est Diff	P-value	T-value	Deg F	Est Diff	P-value	T-value	Deg F	Est Diff	P-value	T-value	Deg F
1	0.00	4.8	0.308	1.03	38	21.1	0.539	0.62	37	-1195.0	**0.000**	-4.83	48
	0.05	-13.3	0.301	-1.05	48	-62.3	0.069	-1.85	56	-955.0	**0.000**	-4.72	39
	0.10	-165.9	**0.015**	-2.55	35	-247.8	**0.003**	-3.15	38	-1019.0	**0.000**	-5.16	36
	0.15	-288.1	**0.001**	-3.45	35	-385.0	**0.001**	-3.78	36	-981.0	**0.000**	-5.27	35
	0.20	-318.0	**0.008**	-2.82	35	-403.0	**0.002**	-3.44	35	-905.0	**0.000**	-4.92	35
2	0.00	5.8	**0.021**	2.40	43	-9.0	0.410	-0.83	56	-1082.0	**0.000**	-4.51	41
	0.05	5.36	**0.015**	2.53	46	-14.8	0.071	-1.84	69	-704.0	**0.000**	-3.92	36
	0.10	5.36	**0.015**	2.53	46	-16.9	**0.020**	-2.39	60	-753.0	**0.000**	-4.36	36
	0.15	5.36	**0.015**	2.53	46	-16.9	**0.020**	-2.39	60	-704.0	**0.000**	-4.71	35
	0.20	5.36	**0.015**	2.53	46	-16.9	**0.020**	-2.39	60	-554.0	**0.000**	-4.48	35
3	0.00	2.8	0.174	1.38	49	-15.08	**0.006**	-2.84	62	-1151.0	**0.000**	-5.12	45
	0.05	3.0	0.172	1.39	47	-14.64	**0.009**	-2.71	64	-1212.0	**0.000**	-6.11	36
	0.10	3.0	0.172	1.39	47	-12.53	**0.014**	-2.53	68	-930.0	**0.000**	-5.41	36
	0.15	3.0	0.172	1.39	47	-12.53	**0.014**	-2.53	68	-772.0	**0.000**	-5.21	36
	0.20	3.0	0.172	1.39	47	-12.56	**0.014**	-2.53	68	-626.0	**0.000**	-537	37

TABLE 1. RESULTS OF TWO SAMPLE T-TESTS FOR INPUTS 1 AND 4 COMPARING CROSSOVER ON VERSUS CROSSOVER OFF. FOR THESE INPUTS, AND ALMOST ALL OTHERS, THE EFFECT OF CROSSOVER IS STATISTICALLY SIGNIFICANT FOR THRESHOLDS *mng_10* AND *mng_5*, THOUGH ITS IMPACT IS NEGATIVE FOR MOST INPUTS.

[14] O. Mengshoel and D. Goldberg, The crowding approach to niching in genetic algorithms. Evolutionary Computation, 16(3), 315-354, 2008.

[15] K. Rodriguez-Vazquez, C. Fonseca, and P. Fleming, Multiobjective genetic programming: A nonlinear system identification application. in 1997 Genetic Programming Conference, 207-212.

[16] K. Sedighi, K. Ashenayi, T. Manikas, R. Wainwright, and H-M. Tai, Autonomous local path planning for a mobile robot using a genetic algorithm. Proceedings of the 2004 IEEE Congress on Evolutionary Computation, 1338-1345.

[17] U. Siddiqi, Y. Shriraishi, and S. Sait, Memory-efficient genetic algorithm for path optimization in embedded systems. IPSJ Transactions on Mathematical Modeling and Its Applications, 6(1), 2013.

[18] N. Srinivas and K. Deb, Multiobjective optimization using nondominated sorting in genetic algorithms. Evolutionary Computation 2(3), 221-248, 1994.

[19] P. Tonupunuri, Evolutionary based path-finding for mobile agents in sensor networks. Master's Thesis, Southern Illinois University, 2008.

[20] G. Yun and H. Lu, Dynamic multiobjective evolutionary algorithm: adaptive cell-based rank and density estimation. IEEE Transactions on Evolutionary Computation, 7(3), 253-274, 2003.

[21] Q. Zhang and H. Li, A multiobjective evolutionary algorithm based on decomposition. IEEE Transactions on Evolutionary Computation, 11(6), 2007.

[22] C. Zheng, M. Ding, C. Zhou, and L. Li, Coevolving and cooperating path planner for multiple unmanned air vehicles. Engineering Applications of Artificial Intelligence, 17, 887-896, 2004.

[23] E. Zitzler and L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength of the Pareto approach. IEEE Transactions on Evolutionary Computation, 3(4), 257-271, 1999.

[24] E. Zitzler, M. Laumanns, and L. Thiele, SPEA2: improving the strength of Pareto evolutionary algorithms. ETH Zurich, 2001.