Enhanced Genetic Path Planning for Autonomous Flight

Vincent R. Ragusa
Florida Southern College
Department of Computer Science
Lakeland, Florida 33801
vragusa@mocs.flsouthern.edu

H. David Mathias
Florida Southern College
Department of Computer Science
Lakeland, Florida 33801
hmathias@flsouthern.edu

Vera A. Kazakova
University of Central Florida
Department of Computer Science
Orlando, Florida 32816
kazakova@cs.ucf.edu

Annie S. Wu
University of Central Florida
Department of Computer Science
Orlando, Florida 32816
aswu@cs.ucf.edu

ABSTRACT

Path planning, the task of finding an obstacle-avoiding, shortest-length route from source to destination is an interesting theoretical problem with numerous applications. We present an improved genetic algorithm for path planning in a continuous, largely unconstrained real-world environment. We introduce a new domain-specific crossover operator based on path intersections. We also implement a new path correction operator that eliminates obstacle collisions from a path, leading to a dramatic search improvement despite the conceptual simplicity of the correction. Finally, in place of a standard binary measure of obstacle collisions, we present a new optimization objective measuring the degree to which a path intersects obstacles. Due to these improvements, individually and in combination, our algorithm is able to solve scenarios that are considerably more complex and exist in a more general environment than those that appear in the literature. We demonstrate the utility of our system through testing onboard an autonomous micro aerial vehicle. Further, our approach demonstrates the utility of domain-specific genetic operators for path planning. We hypothesize that such operators may be beneficial in other domains.

CCS CONCEPTS

•Computing methodologies →Genetic algorithms; Motion path planning;

KEYWORDS

Genetic algorithm, multiobjective optimization, speedup technique, transportation, domain-specific genetic operators

ACM Reference format:
Vincent R. Ragusa, H. David Mathias, Vera A. Kazakova, and Annie S. Wu. 2017. Enhanced Genetic Path Planning for Autonomous Flight. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071293

GECCO '17, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3071178.3071293

Figure 1: Example evolved path for Map 1, shown here in a simulator with satellite imagery.

1 INTRODUCTION

In this paper, we investigate a Genetic Algorithm (GA) approach to path planning for Micro Aerial Vehicles (MAVs). This approach evolves GPS coordinate-based paths onboard an MAV that can then fly the path in a real, obstacle-filled environment, as shown in Figure 1. We examine domain-specific adaptations to the algorithm that we expect to be transferable to other path planning problems and show that leveraging knowledge of the problem can lead to significant performance gain with minimal computational cost.

Path planning refers to the problem of finding a viable route from a source location to a destination within a given environment [16]. The primary algorithmic objective is to optimize some quantity, most often path length. In real world situations, additional objectives may be necessary or desirable. Examples include minimizing the number of waypoints or the sum of the angles of the turns that the robot is required to make [1, 13, 17, 24, 26], maximizing the safety of the resulting path [1, 17], and maintaining a minimum or maximum distance from obstacles or the terrain [13, 24, 26]. Thus, path planning is often a multi-objective optimization problem.

A range of approaches have been applied to path planning. These include both exact and heuristic algorithms [16], with the choice of

type usually dependent on the constraints and goals of the system. *Exact methods* can be computationally expensive but are guaranteed to find a solution or prove that none exists. *Heuristic methods* focus on a more practical perspective, finding the best solution possible within a given time. Specific examples of algorithms that have been studied include, but are not limited to, A* [6, 10, 18], cell decomposition [5], potential fields [2], bug algorithms [3], neural networks [12], and evolutionary algorithms.

Evolutionary algorithms are good candidates for this problem because they have been shown to be capable of evolving paths in both static and dynamic environments, in which obstacle locations are not known in advance or may move [4, 6, 9, 14, 15]. While most GA approaches to path planning are conducted only in simulation [1, 6, 9, 13–15, 17–19, 22, 23], there are examples in which evolved paths are executed on physical robots and vehicles [4, 20, 21].

Limitations of most previous work using GAs for path planning in a manmade environment (one that includes, for example, buildings) are that the environment is discrete, obstacles are axis-aligned quadrilaterals, and motion is monotonic. We remove all of these constraints, allowing for much more realistic and difficult problem instances to be solved. In fact, a dramatic increase in a GAs ability to solve difficult problem instances is one of the main contributions of this work.

Because our algorithm can run on board a real MAV, our approach is guided by two main requirements. First, our system must be able to work in a real-valued rather than discrete environment for compatibility with the standard GPS coordinate system. Second, the limited energy resources on board an MAV means that a fast, efficient algorithm is required.

In previous work, we demonstrate an evolutionary approach to path planning in a complex, continuous, man-made, real-world environment and show that a GA can successfully evolve viable paths for an MAV [20, 21]. We found that a standard crossover implementation was not effective. Related work suggests that incorporating domain-specific information into a GA can improve the performance of the GA search process [9, 15, 24, 26]. We hypothesize that tailor-made genetic operators that take advantage of known problem characteristics may be beneficial in directing the GA exploration effectively. We expect that the benefits of such operators may be transferable to other GAs solving the path planning problem using a similar representation.

We demonstrate that path planning can be greatly enhanced through use of domain-specific features. These features include two novel genetic operators and one new optimization objective. Individually and in combination, these improvements allow our GA to solve significantly more difficult maps and do so with less computation. Our results support the idea that domain-specific knowledge can be used to improve a GA's ability to find a solution.

2 THE PROBLEM

We define path planning as a search for a minimum-length, obstacle-avoiding path P through an environment E, represented as $E(s, e, O)$ where s is a fixed starting point, e is a fixed ending point, and O is a (possibly empty) set of obstacles to avoid. In the work presented here, the only validity criterion is that the number of obstacle collisions is 0. The obstacles specified in a set O define a map.

The environments we consider are two dimensional. This is a necessary concession as the MAV is intended to operate in spaces containing man-made structures. While 2-D building locations are easily found, reliable 3rd dimension (height) information for structures is not readily available. In addition, up-to-date flight regulations may also be difficult to obtain. Thus, all obstacles must be avoided in two-dimensional space, since the algorithm cannot know if it is safe or legal to fly over them. The MAV flies a planned 2-D path, at a fixed altitude, within a 3-D space.

Figure 2 shows the maps for which we present results. These, and other maps on which we have tested our algorithm, are a mix of real-world and contrived scenarios. While real environments demonste the applicability of our system, they are considerably easier to solve than those we devise for increased difficulty. Map 2 requires traversal of multiple narrow corridors connected by 180 degree turns. Maps 3 and 4 are progressively more difficult versions of a zigzag path. Map 3 adds a secondary zigzag that has shorter path length but may be more difficult to discover due to increased distance from the starting point. Map 4 introduces a third, even shorter path that is further still from, and on the opposite side of, the starting point. Significantly, solving maps 3 and 4 requires the algorithm to find a path through the shortest of the possible tracks – even a valid path through a longer track is considered a failure. While our previous work is able to solve maps more difficult than most found in the literature, the new features introduced here are required to reliably solve maps 2, 3, and 4.

When the contents of O are fixed, *i.e.* not updated during the mission, the problem is called *offline* path planning. This is fundamentally different than *online* path planning, in which O may be updated during the mission to include obstacles discovered or moved during the mission. Online path planning involves continuously recalculating the planned path to accommodate the changing model of the environment. The research presented here focuses on offline path planning using a GA. Though offline, our GA runs onboard a MAV allowing new paths to be evolved in the field.

A path P is comprised of starting location s, followed by a sequence of $\ell > 0$ GPS coordinates called waypoints, and terminated with destination e. That is, $P = [s, w_1, w_2, w_3, ..., w_\ell, e]$, where adjacent elements are connected by straight segments. The use of GPS coordinates necessitates the use of real values. In contrast, much of the literature on path planning focuses on algorithms that operate in discrete, finitely subdivided, grid-like environments. Such approaches simplify the path planning problem and are too limited for practical applications, where GPS is the standard for accurate navigation and organically shaped obstacles are commonplace.

There are many possible optimization objectives for path planning. Our baseline algorithm minimizes three objectives: number of waypoints, path length, and number of obstacles hit. The number of waypoints is $|P| - 2$, denoted ℓ in the path definition above. Minimizing waypoints has two benefits: (1) simplifying the mission and (2) reducing runtime as there is a dependence on the number of path segments. Path length is defined as:

$$len = d(s, w_1) + d(w_\ell, e) + \sum_{i=1}^{\ell-1} d(w_i, w_{i+1})$$

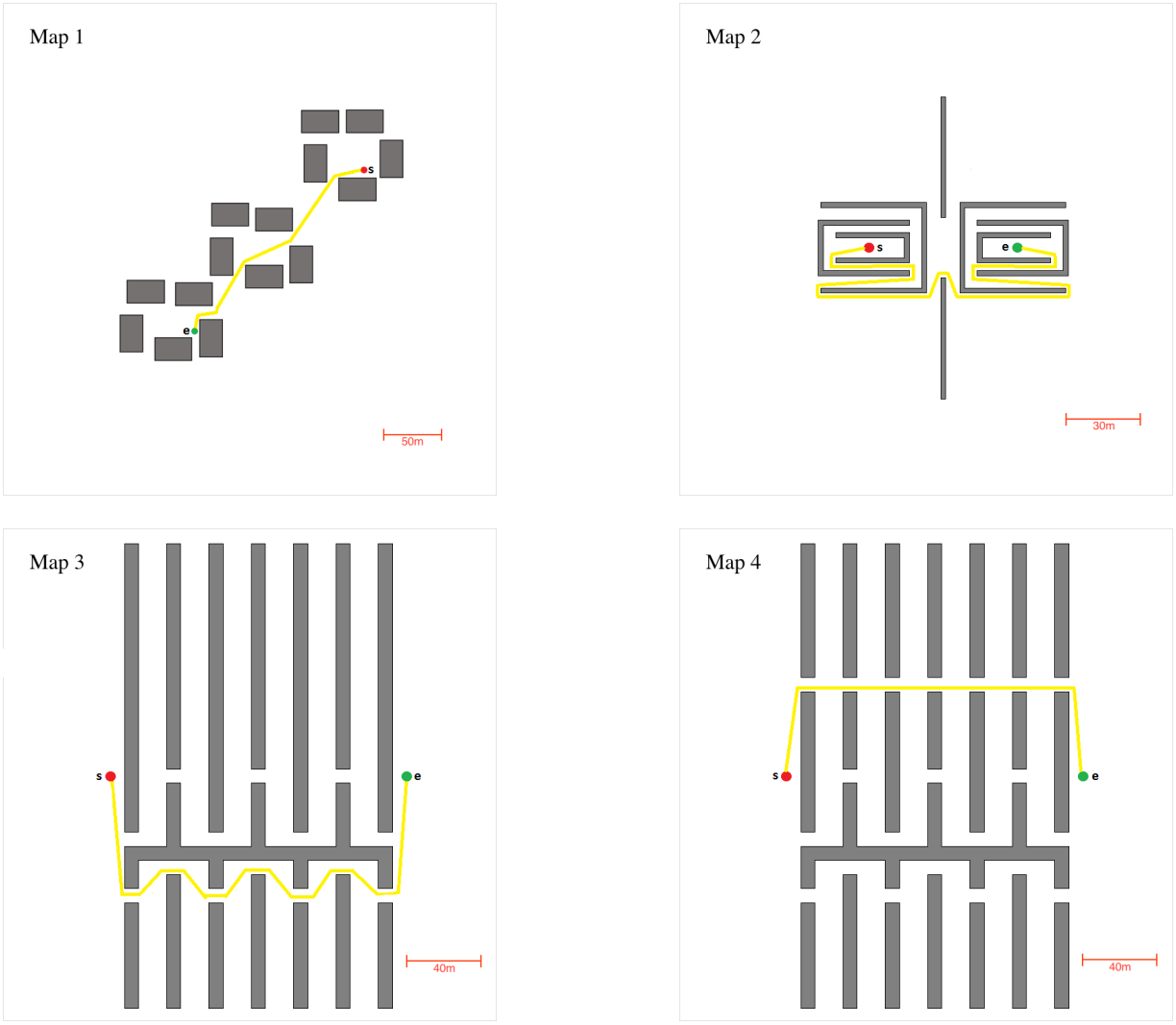

Figure 2: Environments for which results are reported. Map 1 represents actual buildings on a large college campus. Maps 2, 3, and 4 were contrived to be difficult to solve within the path length threshold. On each map, a path approximating the optimal path is shown in yellow. We note that due to the distance threshold, only runs that find a path with length within 5% of optimal are considered successful. This limits success to those paths that use the same passage (or, in the case of Map 2, one that is symmetric) through the obstacles as the optimal path.

where $d(x, y)$ is the Euclidean distance from x to y. The number of obstacles hit is simply a count of the obstacles intersected by at least one path segment.

3 THE ALGORITHM

In this section, we describe our base algorithm, which is derived from the well-known NSGA-II by Deb *et al* [8]. Although there are more recent multi-objective evolutionary algorithms [7, 25], NSGA-II is straightforward to implement, appropriate for the number of objectives in our problem, and reasonably efficient. To allow for maximum flexibility in addressing domain-specific aspects of path planning, we implemented the algorithm rather than use an existing version such as DEAP [11].

A path consists of a sequence of *waypoints*, each a GPS coordinate to which the MAV will travel. A genome corresponds to a path represented by a list of real-valued coordinate pairs, $(latitude, longitude)$. For applied path planning algorithms, a real-valued genome is better-suited than a binary representation as it allows for finer control of path evolution by the genetic operators.

For the population at generation 0, each member is created at random with 2 to 6 waypoints. A *geo fence* parameter constrains each waypoint to be within a specified Euclidean distance of the midpoint between the starting and ending points. The purpose of

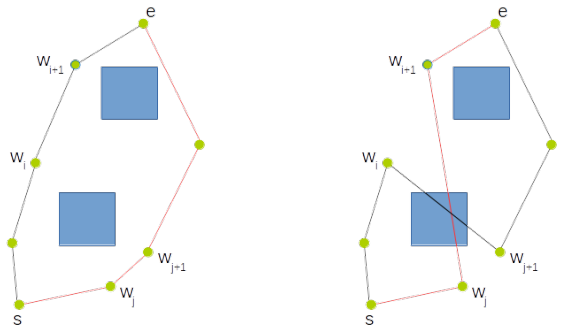

Figure 3: Depiction of crossover version XO-R, in which cutpoint positions are chosen at random.

this value is to mirror the geo fence within the firmware in the MAV's flight control board.

Let pop_x represent the population at generation g_x. In the base algorithm, population pop_{x+1} is created from population pop_x using a combination of mutation and single-point crossover, which we denote XO-R due to random selection of the cut point. Figure 3 illustrates the effect of XO-R. Children created are placed in child population pop_c. The algorithm produces children until $|pop_c| = |pop_x| = n$ at which time pop_x and pop_c are merged into a combined population pop with size $2n$. pop_{x+1} is obtained from pop using standard selection based on non-dominated sorting [8].

Crossover events occur with probability pop_{xo}. The base algorithm uses single-point crossover. Due to the genome structure, cut points must lie between waypoints and cannot occur before s or after e. Parents are chosen in a binary tournament. The cut point in each parent is chosen independently as genome lengths can vary.

With probability $1 - p_{xo}$ a mutation event occurs instead of a crossover event. In this case, a single parent is chosen by binary tournament. The parent is copied and the copy mutated. We implement four mutation operators, *add, delete, swap,* and *move. add, delete,* and *move* each affect a single, randomly selected waypoint while *swap* exchanges the position within the genome of two adjacent waypoints, also selected at random. When a member is chosen for mutation, exactly one of the operators is applied. The probabilities of the operators are 0.25, 0.15, 0.1 and 0.5, respectively. Mutations are subject to the geo fence parameter described above.

4 IMPROVEMENTS

In this section, we detail two new genetic operators and one new objective that offer a significant decrease in the number of generations required to find solutions, as well as a dramatic increase in the complexity of maps the algorithm can solve successfully.

4.1 Crossover Versions

Though typical for genetic path planning algorithms, the single-point crossover described in Section 3 is largely ineffective [20]. This is, perhaps, not surprising given that the genome represents spatial information but there is no spatial basis for choosing cut locations or applying crossover. To address this, we introduce a

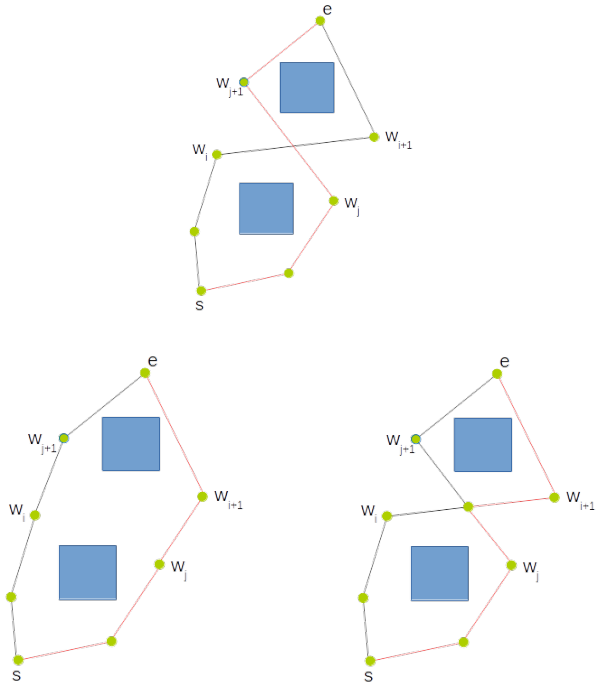

Figure 4: Depiction of intersection crossover. The top image shows two intersecting paths. The lower left and right show the result using XO-I and XO-I$^+$, respectively.

new form of crossover specific to path planning, which we call *intersection crossover*. This operator allows crossover only between genomes representing paths that intersect within the environment.

In intersection crossover, parent selection occurs via tournament selection as for XO-R. Once parents have been chosen, crossover occurs only if the paths they encode intersect. Otherwise, rather than attempt multiple times to choose intersecting paths, each parent is copied and the copies mutated.

We implement two versions of intersection crossover: (1) XO-I and (2) XO-I$^+$. To define them formally, let p_1 and p_2 be members of the population with genomes $\langle s, w_{0_1}, \ldots, w_i, w_{i+1}, \ldots, w_{\ell_1}, e\rangle$ and $\langle s, w_{0_2}, \ldots, w_j, w_{j+1}, \ldots, w_{\ell_2}, e\rangle$, respectively. Further, let p_1 and p_2 intersect at point y on path segments $\langle w_i, w_{i+1}\rangle$ and $\langle w_j, w_{j+1}\rangle$.

Figure 4 illustrates XO-I and XO-I$^+$. In XO-I, child $c_1 = \langle s, w_{0_1}, \ldots, w_i, w_{j+1}, \ldots, w_{\ell_2}, e\rangle$ and child $c_2 = \langle s, w_{0_2}, \ldots, w_j, w_{i+1}, \ldots, w_{\ell_1}, e\rangle$. Informally, c_1 includes the waypoints of p_1 from s up to w_i, the waypoint immediately before intersection point y, and the waypoints of p_2 from w_{j+1}, the waypoint immediately after y, to e.

XO-I$^+$ augments each child with intersection point y. This yields children $c_1 = \langle s, w_{0_1}, \ldots, w_i, y, w_{j+1}, \ldots, w_{\ell_2}, e\rangle$ and $c_2 = \langle s, w_{0_2}, \ldots, w_j, y, w_{i+1}, \ldots, w_{\ell_1}, e\rangle$.

4.2 Obstacle Intrusion

The standard objective related to obstacle avoidance is a simple count of the number of obstacles hit. Minimizing this quantity to 0 yields a collision-free path; however, this measure of collisions

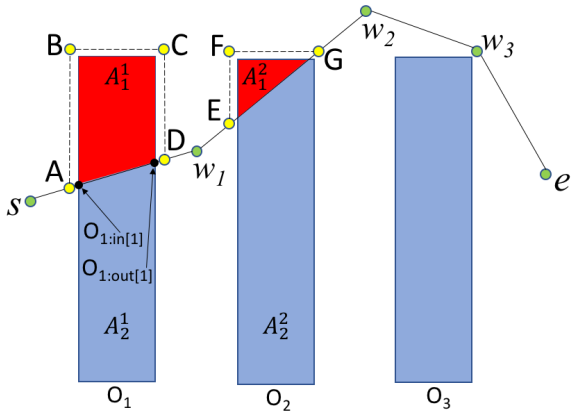

Figure 5: The obstacle intrusion measure and path correction operator use the same information in different ways. Consider path $p = \langle s, w_1, w_2, w_3, e\rangle$ through this simple map. The value of the obstacle intrusion objective for this input/path pair is $A_1^1 + A_1^2$. Applying path correction to p results in path $p' = \langle s, A, B, C, D, w_1, E, F, G, w_2, w_3, e\rangle$.

has a shortcoming: a path that crashes through the middle of an obstacle is indistinguishable (with respect to that objective) from one that only slightly clips a corner of an obstacle. This limits the evolutionary utility of the obstacle avoidance objective.

We present an alternative, real-valued measure of collision, called *obstacle intrusion*, based on the degree to which a path intersects an obstacle. The resulting finer fitness granularity improves the algorithm's ability to distinguish and reward degrees of obstacle intrusion, thus facilitating evolutionary search.

The *obstacle intrusion* value is calculated as follows. The points at which the path enters and exits an obstacle $o_j \in O$ are determined. As there can be multiple enter/exit pairs for o_j, we denote the i-th pair as $pair_i = (o_{j:in[i]}, o_{j:out[i]})$. Connected by the path segments between them, each pair divides the obstacle into two regions with areas A_1^i and A_2^i. The obstacle intrusion value for $pair_i$ on obstacle o_j is $I_j^i = min(A_1^i, A_2^i)$. The total obstacle intrusion value for o_j is $I_j = \sum_i I_j^i$ and the total intrusion value for a population member as $\sum_{j \in |O|} I_j$, the sum over all obstacles. See Figure 5.

It would likely improve evolution to have a separate objective for each obstacle in the environment. However, this introduces implementation problems as the number of objectives would be both variable and large. While the objective as presented here is a compromise in this sense, it proves extremely beneficial in testing.

4.3 Path Correction

The *path correction* operator eliminates obstacle collisions by "pulling" a path off of any obstacles it hits. Consider again the points at which a path enters and exits an obstacle. The algorithm determines the shortest path, along the obstacle perimeter, between these two points. It then replaces in the genome the path segments between entry and exit with a new subpath following the shortest path along the perimeter, translated 1m outside the perimeter. Figure 5 shows the effect of path correction.

If used, path correction can be applied once, to the initial population, or repeatedy at some fixed interval. When path correction is applied to the initial population, we denote it IPC. When it is applied at a fixed interval of x generations, we denote it RPC-x. RPC-0 denotes the case in which path correction is not applied repeatedly. We hypothesize that applying path correction too frequently might inhibit evolution by limiting exploitation.

5 EXPERIMENTAL SETUP

The experiments we describe are designed to evaluate the effectiveness of the new operators and optimization objective we introduce. Each experiment consists of 80 trials where each trial includes runs with each of the combinations of parameters being tested. Trials are run for all four maps. Within a trial, the initial population is the same for each run to eliminate a source of variability. Experiments are run on the Stampede supercomputer at the Texas Advanced Computing Center. Each standard compute node on Stampede includes two Intel Xeon E5-2680 8 core Sandy Bridge processors and 32GB of RAM.

The values of several parameters used in this work were established empirically in prior works [20, 21]. These include the probabilities with which the four mutation operators are applied (0.25, 0.15, 0.1, 0.5, respectively), and the post-crossover mutation probability (0.0). In addition, the utility of the waypoint count objective is found to be somewhat beneficial. Other parameter values are established empirically as part of this work, including the crossover probability (0.4) and whether path correction is applied to generation 0 (no).

As discussed in Section 2, successful path planning requires satisfying the validity conditions as well as minimizing, to the extent possible, the optimization objectives. Given our desire to improve the efficiency of path planning, we also examine the number of generations required to find a valid path of acceptable length. As unsuccessful runs are expensive, we measure the success rates of the trials. In this context, success is defined as finding a path within 5% of optimal in fewer than a specified maximum number of generations. For these experiments the maximum generations is 3000. Within the set of successful runs, we track the mean number of generations and the standard deviation.

Table 1 summarizes our sequence of experiments. We conduct the experiments in sequence so that once we establish the utility of a feature, it can be used, or excluded, to enhance overall performance in subsequent experiments. Thus, experiments listed lower in Table 1 include most or all of the features described.

6 RESULTS & DISCUSSION

For each map tested, we establish an optimal path length. A run is considered successful if it finds a valid path with a length within 5% of optimal. The success rate is the percentage of runs that are successful. For each experiment, we report the success rate (S-R), the mean number of generations required for success (S-M), and the standard deviation in the number of generations (S-SD).

6.1 Effects of obstacle intrusion

To determine the impact of obstacle intrusion, we run trials that isolate it from the path correction operator. For the maps tested, we present strong evidence that all aspects of the algorithm's performance benefit from employing this new objective. Specifically,

Feature	Values Tested	Explanation	Other Parameters	Results
obs_intrusion	*off, on*		*RPC-0, XO-I$^+$*	Table 2
path_correction	*RPC-0* *IPC* *RPC-100* *IPC ∧ RPC-100*	*no path correction* *path correct initial population* *path correct every 100 generations* *path correct initial pop & every 100 gens*	*obs_intrusion*	
correction_interval	*RPC-1, RPC-5, RPC-20, RPC-50, RPC-100*	*path correction every X generations*	*obs_intrusion, XO-I$^+$, [IPC: off, on]*	Figure 6
crossover	*XO-Off* *XO-R* *XO-I* *XO-I$^+$*	*no crossover* *random cutpoint crossover* *intersection crossover* *intersection crossover, point added*	*obs_intrusion, [RPC-0, RPC-20]*	Table 3
avg_move	*8, 12, 16, 24, 32, 40, 48, 64*	*average distance of move mutation*	*obs_intrusion, RPC-20, [XO-Off, XO-R, XO-I, XO-I$^+$]*	

Table 1: Overview of the experiments reported. The first and second columns list the feature that is the primary subject of the experiment and the values being tested. The 4th column provides values for other relevant parameters. As indicated, some experiments used all new features in concert. The last column provides pointers to where data can be viewed.

	Success Rate (S-R)		Success Mean (S-M)		Success SD (S-SD)	
	Off	**On**	**Off**	**On**	**Off**	**On**
Map 1	37.50%	100.00%	903.86	162.23	870.03	76.62
Map 2	0.00%	67.42%	N/A	1473.57	N/A	678.06
Map 3	0.00%	82.11%	N/A	831.99	N/A	388.29
Map 4	1.09%	77.17%	12.00	71.21	N/A	41.75
Average	9.65%	81.67%	457.93	634.75	870.03	296.19

Table 2: Effect of obstacle intrusion on all maps.
Comparison of runs on maps 1 through 4 with obstacle intrusion off and on. Runs were performed without path correction and with XO-I$^+$. S-M and S-SD are presented as number of generations.

success rate increases dramatically, mean generations decrease, standard deviations of generations for successful runs decrease, and the percentage of successful runs drastically increases.

In particular, only the least complex map, map 1, has significant success (37.5%) without obstacle intrusion. Maps 2, 3, and 4 have success rates of 0%, 0%, and 1.09%, respectively. The 1.09% value for map 4 represents a single successful run. With obstacle intrusion enabled, the success rates jump to 100%, 67.42%, 82.10%, and 77.17%, respectively.

When examining the mean number of generations, there is a seemingly negative effect of obstacle intrusion on map 4. This is due to an outlying event where a single success is achieved extremely fast while obstacle intrusion is off. Thus, the mean generation measure is artificially low for the obstacle intrusion case. This behavior, however, is not representative of having obstacle intrusion turned off for said map, as evidenced by the 1% success rate, as opposed to the 77% success rate obtained when obstacle intrusion is on. Table 2 summarizes our results.

We further analyze the cost of obstacle intrusion in terms of actual runtimes. The goal is to ensure that a decrease in the number of mean generations is not offset by a commensurate increase in the time to evolve each generation. With obstacle intrusion on, each generation can take up to 3 times as long as generations with obstacle intrusion off. However, since the success rates increase dramatically and the overall number of generations decreases, the additional time per generation is warranted. Only map 1 has sufficiently high success rates in both cases to allow meaningful comparison. Average time for successful runs with obstacle intrusion off is 170.85 versus 50.32 seconds when the objective is on.

6.2 Effects of path correction

Figure 6 summarizes the results of the path correction tests. The graph shows that, on the less deceptive maps (namely, maps 1 and 2), most path-correction interval setups succeed in all cases (light blue and black lines) with the exceptions of "never" (RPC-0) and "only on the initial generation"(IPC & RPC-0). Nevertheless, the number of generations required to find a successful path does vary with parameters. On these maps, path-correcting more often tends to lead to improved results; however, the differences are modest.

On the more deceptive maps (*i.e.* maps 3 and 4), path-correction setups have a stronger distinguishing effect (note the differences among the yellow and green bars across the path-correction intervals). Specifically, not validating the initial population appears to allow for increased initial exploration, an effect that is strengthened when the path-correcting interval is larger, thus increasing the initial exploration period (note the generally lower bars on the left-most set of RPC-1 – RPC-100 intervals, versus the taller bars on the right-most set of IPC & RPC-1 – RPC-100 in figure 6). Based on these findings, it appears that waiting to path-correct the population is likely to lead to better exploration of the solution space. In our experiments, best overall performances, as measured by a combination of high success rate and low average number of generations, are obtained by the setups of path-correcting every 20 or every 50 generations, coupled with NOT path-correcting the initial population.

Additionally, when comparing path-correction intervals against runs with no path-correction (RPC-0), we see that the difference can be stark but is map specific. For example, while only a small difference is observed on map 1 (compare first dark blue bar to the others), map 2 strongly benefits from path-correcting at any interval (compare the first red bar on the graph to the others). On maps 3 and 4, however, the benefit of path-correction isn't as clear. In fact, many setups lead to severely decreased success rates. Nevertheless, among the beneficial setups we observe an increase in the number of successful runs and a decrease in the number of generations and even standard deviations.

While future testing will include a variety of larger maps, real world scenarios are generally likely to be less deceptive than the scenarios tested in this work (closer to map 1, for instance). Nevertheless, since the specifics of the deceptiveness may vary, future algorithm improvements would likely revolve around increasing its adaptability to the unpredictable nature of the map. Since it is

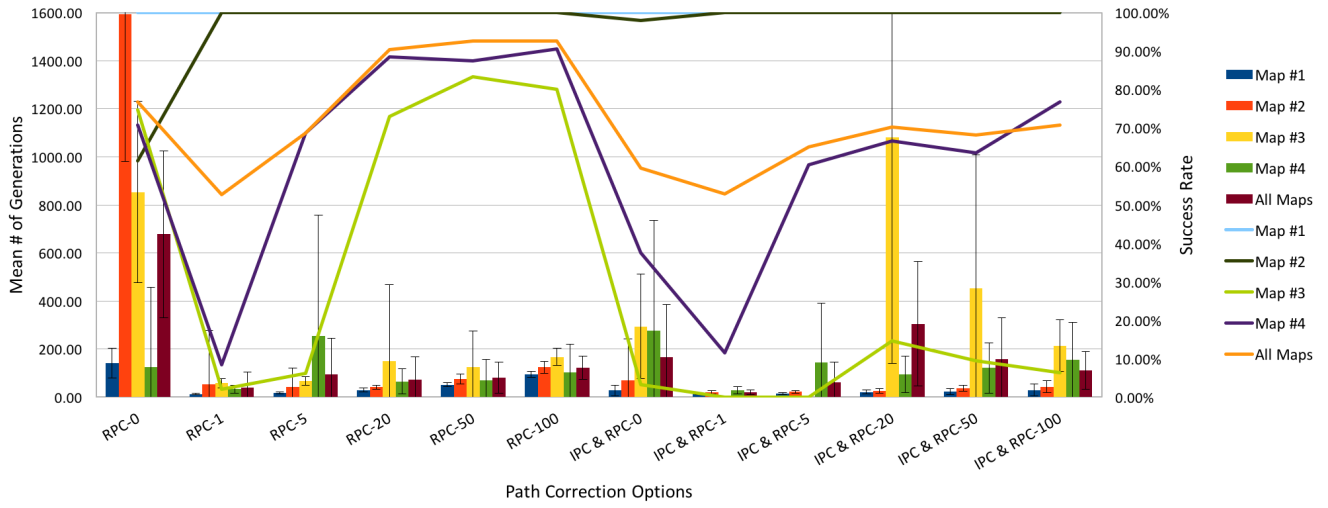

Figure 6: **Results for experiments on the frequency with which path correction is applied. IPC indicates that members of the initial population were path corrected. Obstacle intrusion was on in all cases. The lines and scale on the right indicate success rates for each of the four maps.**

	XO-Off		XO-R		XO-I		XO-I$^+$	
	RPC-0	RPC-20	RPC-0	RPC-20	RPC-0	RPC-20	RPC-0	RPC-20
	Map 1							
S-R	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
S-M	503.60	38.09	337.85	35.25	259.28	31.96	163.66	32.18
S-SD	255.55	7.98	193.14	5.81	128.67	6.85	75.29	7.09
	Map 2							
S-R	0.00%	99.87%	0.00%	100.00%	0.00%	100.00%	59.00%	100.00%
S-M	N/A	79.75	N/A	71.42	N/A	43.48	1501.58	46.90
S-SD	N/A	65.00	N/A	17.20	N/A	9.39	611.63	15.31
	Map 3							
S-R	4.26%	85.16%	1.06%	82.55%	52.13%	87.89%	69.15%	80.99%
S-M	2250.25	184.03	2898.00	259.66	2126.94	136.69	734.38	160.24
S-SD	491.19	282.42	0.00	474.73	528.17	282.65	280.95	345.63
	Map 4							
S-R	71.88%	90.89%	80.21%	82.55%	82.29%	86.98%	69.79%	85.81%
S-M	73.20	65.32	287.22	259.66	86.63	44.79	130.31	58.83
S-SD	34.08	77.15	581.00	474.73	228.90	45.55	374.46	94.83
	Across Four Maps							
S-R	44.03%	93.98%	45.32%	92.32%	58.61%	93.72%	74.49%	91.70%
S-M	942.35	91.79	1174.36	104.97	824.28	64.23	632.48	74.54
S-SD	260.27	108.13	258.05	143.30	295.24	86.11	335.58	115.72

Table 3: **Crossover comparisons across four maps. Comparison of no path-correction (RPC-0) versus path correction every 20th generation, without validating the initial population (RPC-20), across mutation distances centered at 8, 12, 16, 24, 32, 40, 48, and 64 meters. S-R, S-M, and S-SD represent success rate and the mean and standard deviation, in number of generations, for successful runs.**

generally advisable to conduct multiple evolution runs, one sensible option is to employ various path-correction intervals across these runs to mediate the impossibility of knowing the specifics of the sought after path before it is found by the GA.

6.3 Effects of crossover versions

Table 3 compares the effects of having no crossover versus standard random cut-point crossover (XO-R) versus two new crossovers that employ path intersections as the basis for defining cut points (XO-I and XO-I$^+$).

When looking at averages across all maps, we see that XO-R leads to a comparable success rate as having no crossover, while increasing both the number of generations to find a solution as well as the standard deviation. These findings, along with the additional cost of performing crossover, suggest that no crossover is preferred to XO-R.

When comparing no crossover to XO-I and XO-I$^+$, the new crossovers have a definite advantage. Both new operators have comparable success rates, coupled with a decreased number of generations and standard deviation. On a map by map basis, the advantages are not obvious on maps 1 and 2, but become more apparent on maps 3 and 4, although no clear winner emerges. The superiority of the new crossovers stems from their leveraging of the existing intersection points, which indicate the crossing paths are spatially compatible and could exchange segments before and after the intersection point. If paths do not already intersect, swapping random segments is likely to be strongly disruptive to the formed solutions, complicating exploitation of evolved sub-paths.

6.4 Effects of mutation step size

To further verify the observed crossover differences, a number of different mutation sizes are tested (mutation distances centered at 8, 12, 16, 24, 32, 40, 48, and 64 meters). When looking at setups with each of the different mutation sizes for each of the four crossover types (no crossover, XO-R, XO-I, and XO-I$^+$), we see no clear winning distance across all maps. While it is to be expected that the ideal average path mutation distance is map specific, we are able to observe a general trend: evolution behavior generally improves as mutation distance increases, approaching some map-specific value, and decreases again as this value is surpassed.

As before, on maps 1 and 2 no significant differences are observed upon increasing mutation distances. On map 4, higher mutation distances appear to be beneficial. We believe this is a result of a straight segment being part of the sought after, but more difficult

to find, ideal path. On Map 3, increasing average mutation size beyond 40 m is no longer beneficial, which can be explained by the map's lack of a straight path as found in map 4.

Based on our findings, an average mutation of 24-32 meters appears to work well for the tested scenarios. For the obstacle spacing and shapes present, these distances resulted in an effective compromise between exploring the space and exploiting solution features. Given that beneficial mutation distances are map specific, it would be sensible to incorporate dynamically adjusting mutation distances. Note that since no winning mutation size emerged, crossover comparison values presented in Table 3 are averaged across all of the tested mutation distances (8 through 64 meters).

7 CONCLUSIONS

In this work, we present a GA for path planning in obstacle-filled, real-world environments and show that incorporating domain knowledge into the genetic operators can benefit GA performance on this problem. We present two new domain-specific genetic operators and a new domain-specific optimization objective that dramatically improve the success rate of the algorithm and the number of generations required to achieve success. Further, the maps for which the algorithm can successfully find paths are significantly more complex and difficult to solve than in previous work.

In future work, we would like to conduct experiments to determine if using XO-I and XO-I$^+$ in concert proves beneficial. This is motivated by the data showing that for some maps XO-I is significantly better than XO-I$^+$; for others, the situation is reversed. A future goal is to find a way to leverage the benefits of both crossover versions within a single run.

Some parameters, such as the move mutation distance and the path correction frequency, demonstrate utility for all maps but to varying degrees. For example, the algorithm performs better with path correction every 50 generations for Map 3, but every 20 generations for other maps. We speculate that it might be possible to leverage features in the environment or observed rates of change in evolutionary progress to adapt these parameter values dynamically.

Another useful future direction might be to explore improvements to obstacle intrusion. For example, it may be possible to maintain this objective value separately for each obstacle without creating an unmanageably large number of objectives. Further, the current implementation of obstacle intrusion is complex. It may be possible to simplify the implementation and, in the process, make the algorithm more efficient.

ACKNOWLEDGMENTS

The authors thank the National Science Foundation Extreme Science and Engineering Discovery Environment (XSEDE) for supporting this work through availability of computing resources. We also thank Peter Barker for assistance with the flight simulator.

REFERENCES

[1] Faez Ahmed and Kalyanmoy Deb. 2011. Multi-Objective Path Planning using Spline Representation. In *Proc. IEEE Int'l Conf. Robotics & Biomimetics*. 1047–1052.
[2] K. S. Al-Sultan and M. D. S. Aliyu. 1996. A new potential field-based algorithm for path planning. *Journal of Intelligent & Robotic Systems* 17, 3 (1996), 265–282.
[3] N. Buniyamin, W. Wan Ngah, N. Shariff, and Z. Mohammad. 2011. A simple local path planning algorithm for autonomous mobile robots. *Journal of Systems Applications, Engineering & Development* 5, 2 (2011), 151–159.
[4] H. Burchardt and R. Salomon. 2006. Implementation of path planning using genetic algorithms on mobile robots. In *Proc Congress on Evol Comp*. 1831–1836.
[5] H. Choset and P. Pignon. 1997. Coverage path planning: the boustrophedon decomposition. In *International Conference on Field and Service Robotics*.
[6] Valeria de Carvalho Santos, Claudio Fabiano Motta Toledo, and Fernando Santos Osorio. 2015. An Exploratory Path Planning Method based on Genetic Algorithm for Autonomous Mobile Robots. In *IEEE Congress on Evolutionary Computation*.
[7] K. Deb and H. Jain. 2014. An evolutionary many-objective optimization algorithm using reference-point-based non-dominated sorting approach, Part I: Solving problems with box constraints. *IEEE Trans. Evol. Comp.* 18, 4 (2014), 577–601.
[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans Evol. Comp.* 6, 2 (2002), 182–197.
[9] Ahmed Elshamli, Hussein A. Abdullah, and Shawki Areibi. 2004. Genetic algorithm for dynamic path planning. In *Proc. Canadian Conf Elec. & Comp. Engr.*
[10] D. Ferguson, M. Likachev, and A. Stentz. 2005. A guide to heuristic-based path planning. In *Proc. Int'l Conf. on Automated Planning and Scheduling*.
[11] F-A. Fortin, F-M. De Rainville, M-A. Gardner, M. Parizeau, and C. Gagne. 2012. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research* 13 (July 2012), 2171–2175.
[12] R. Glasius, A. Komoda, and S. Gielen. 1995. Neural Network dynamics for path planning and obstacle avoidance. *Neural Networks* 8, 1 (1995), 125–133.
[13] I. Hasircioglu, H. Topcuoglu, and M. Ermis. 2008. 3-D path planning for the navigation of unmanned aerial vehicles by using evolutionary algorithms. In *Genetic Algorithms and Evolutionary Computation Conference*. ACM, 1499–1506.
[14] A. Hermanu, T. Manikas, K. Ashenayi, and R. Wainwright. 2004. Autonomous robot navigation using a genetic algorithm with an efficient genotype structure. In *Intelligent Engineering Systems Through Artificial Neural Networks: Smart Engineering Systems Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Complex Systems and Artificial Life*. ASME Press.
[15] Cem Hocaoglu and Arthur C. Sanderson. 2001. Planning Multiple Paths with Evolutionary Speciation. *IEEE Trans. Evol. Comp.* 5, 3 (2001), 169–191.
[16] Yong K. Hwang and Narendra Ahuja. 1992. Gross motion planning – A survey. *Comput. Surveys* 24, 3 (1992), 219–291.
[17] H. Jun and Z. Qingbao. 2010. Multi-objective mobile robot path planning based on improved genetic algorithm. In *IEEE International Conference on Intelligent Computation Technology and Automation*. 752–756.
[18] Shidong Li, Mingyue Ding, and Chao Cai. 2009. A novel path planning method based on path network. In *6th Int'l Symp Multispectral Image Processing and Pattern Recognition*.
[19] H.-S. Lin, J. Xiao, and Z. Michalewicz. 1994. Evolutionary navigator for a mobile robot. In *International Conference on Evolutionary Computation*. IEEE, 2199–2204.
[20] D. Mathias and V. Ragusa. 2016. An empirical study of crossover and mass extinction in a genetic algorithm for pathfinding in a continuous environment. In *Proceedings of Congress on Evolutionary Computation*. IEEE.
[21] D. Mathias and V. Ragusa. 2016. Micro aerial vehicle pathfinding and flight using a multi-objective genetic algorithm. In *Proc. Intelligent Systems Conf.*
[22] K. Sedighi, K. Ashenayi, T. Manikas, R. Wainwright, and H-M. Tai. 2004. Autonomous local path planning for a mobile robot using a genetic algorithm. In *Proceedings of Congress on Evolutionary Computation*. IEEE, 1338–1345.
[23] Jianping Tu and Simon X. Yang. 2003. Genetic algorithms based path planning for a mobile robot. In *Proc. IEEE Int'l Conf Robotics & Automation*. 1221–1226.
[24] J. Xiao, Z. Michalewicz, L. Zhang, and K. Trojanowski. 1997. Adaptive evolutionary planner/navigator for mobile robots. *IEEE Trans Evol Comp* 1, 1 (1997).
[25] Q. Zhang and H. Li. 2007. A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation* 11, 6 (2007).
[26] C. Zheng, M. Ding, C. Zhou, , and L. Li. 2004. Coevolving and cooperating path planner for multiple unmanned air vehicles. *Engineering Applications of Artificial Intelligence* 17 (2004), 887–896.