Ant Colony Optimization for Agile Motion Planning

by

Tom Krenzke

B.S. Aerospace Engineering
University of Illinois at Urbana-Champaign (2004)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2006

Author . . . .
Department of Aeronautics and Astronautics
May 12, 2006

Certified by . . . .
Dr. Marc McConley
CSDL Technical Supervisor
Thesis Supervisor

Certified by . . . .
Dr. Brent Appleby
Lecturer in Aeronautics and Astronautics
Thesis Supervisor

Accepted by . . . .
Dr. Jaime Peraire
Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

Ant Colony Optimization for Agile Motion Planning

by

Tom Krenzke

Submitted to the Department of Aeronautics and Astronautics
on May 12, 2006, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

With the need for greater autonomy in unmanned vehicles growing, design of algorithms for mission-level planning becomes essential. The general field of motion planning for unmanned vehicles falls into this category. Of particular interest is the case of operating in hostile environments with unknown threat locations. When a threat appears, a replan must be quickly formulated and executed. The use of terrain masking to hide from the threat is a vital tactic, which a good algorithm should exploit. In addition, the algorithm should be able to accommodate large search spaces and non-linear objective functions. This thesis investigates the suitability of the Ant Colony Optimization (ACO) heuristic for the agile vehicle motion planning problem. An ACO implementation tailored to the motion planning problem was designed and tested against an existing genetic algorithm solution method for validation. Results show that ACO is indeed a viable option for real-time trajectory generation. ACO's ability to incorporate heuristic information, and its method of solution construction, make it better suited to motion planning problems than existing methods.

Thesis Supervisor: Dr. Marc McConley
Title: CSDL Technical Supervisor

Thesis Supervisor: Dr. Brent Appleby
Title: Lecturer in Aeronautics and Astronautics

Acknowledgments

my acknowledgments
must thank those that got me through
why not try haiku
thesis nearly complete
thoughts of sunshine drive me on
must write like the wind

This thesis would not have been possible were it not for the support I received along the way. First, I would like to thank my fellow Draper Fellows who were always willing to help me brainstorm ideas, or just chat to kill some time until lunch rolled around. Brian, Stephen, Drew, John, Joe, Larry, Peter Hans, and all the rest. Thanks to my advisors, Brent Appleby and Marc McConley, for their guidance.

Finally, I would like to thank Chuck Norris. His hard-hitting style of Texas justice was a constant source of inspiration. Also, I'm afraid of what he'd do to me if I didn't put him in the Acknowledgments.

It has been quite a journey, and the thing that I'll remember most fondly are all the friends I've made along the way.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The field of unmanned aerial vehicles (UAVs) has made tremendous strides in recent years. With the wide range of shapes and sizes of UAVs available, an equally broad range of applications becomes possible. Without the systems in place to support a human pilot, UAVs can have greater endurance and agility than manned aircraft. Commercial applications include cargo transportation and weather monitoring. These are tasks that a human pilot might become fatigued performing, due to the tedious nature of the operation. A UAV would not be prone to this problem. Military applications such as reconnaissance and target monitoring also seem well-suited to unmanned aircraft. Using UAVs in hazardous environments has the benefit of reducing risk to human pilots. Team missions in which unmanned vehicles are led by a manned vehicle are also a possibility. Coordination of a team mission by a small group of operators would require that each agent be able to act independently in order to remove unnecessary operator fatigue.

Several examples of operational UAVs include the Predator, Global Hawk, and UCAV. Both the Predator and Global Hawk have demonstrated their capability in warfare. Since 1995 the Predator has flown hundreds of missions and thousands of operational hours. Since 2001 the Global Hawk has flown over 50 missions and 1000 hours. Global Hawk also set world records for UAV endurance, flying 7500 miles nonstop across the Pacific to Australia. The Boeing UCAV is expected be the most technologically advanced and exhibit the greatest deal of autonomy. It has demonstrated its ability to do inflight replans during test flights [25]. More advances are certain to appear in the years to come. It is predicted that over a dozen fully operational UAV programs will exist by 2013.

Before UAVs can reach their full potential and replace human pilots, a greater level of autonomy is needed. The current level of autonomy in unmanned systems has been described as 'supervisory control' [15, 23]. That is, a human operator (or several) are continually in the loop and aiding the UAV in decision making. The human is generally responsible for setting up high-level tasks such as waypoint following, and then monitoring the UAV as it executes the plan. Mission replanning

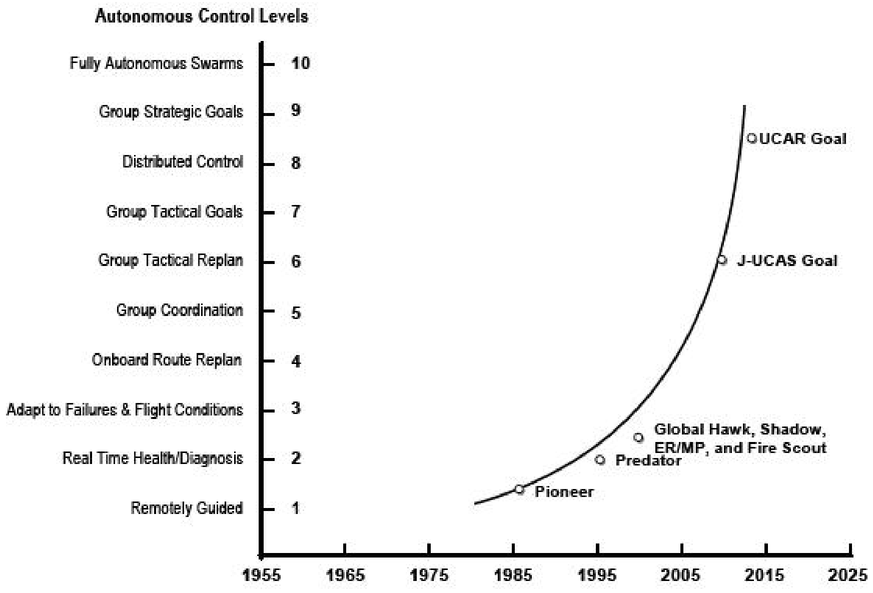


Figure 1-1: Trend in UAV Level of Autonomy [23]

capabilities by the UAV are limited. To be effective in military applications, the UAV must be able to intelligently make mission-level decisions in response to new information. Especially when operating in hazardous environments the UAV must be able to operate without input from a ground station. This increased level of autonomy has several benefits. First, loss of communications between the UAV and any ground station due to distance or interference will not result in a mission-ending failure. Second, as the UAV is able to make more decisions on its own the operator's workload will be reduced. Third, in the case of the appearance of an unexpected threat the UAV might not have time to inform the ground operator and wait for a course of action to be sent. Instead, it should be able to quickly formulate and execute an escape route. Generating trajectories offline for the entirety of the mission in a partially known environment is problematic as the plan may be invalidated by detection of a threat. As such, online replanning is essential.

1.2 Objective

The goal of this thesis is development and validation of a real-time motion planner for use in agile aerial vehicles. In particular, application to the case of operating in threat-laden environments in which quick response to previously unknown threats is needed. The optimization paradigm used is that of Ant Colony Optimization (ACO). An ACO implementation was created specifically for the agile motion planning problem. In order to validate the performance of this method, a currently existing method was used as a benchmark. The benchmark used was a Genetic Algorithm (GA) tailored to the same problem. Using this existing method as a benchmark is reasonable due to the similarities between ACO and GA. Also, a great deal of time and energy was put into the GA's development, so outperforming it is a non-trivial objective. It should be noted that the goal of this algorithm is to generate a 'good' solution quickly. The real-time aspect of this algorithm is a critical concern. High-quality trajectories should be generated in under a few seconds.

ACO should perform better for optimal agile maneuver planning as a function of time due primarily to the way in which ACO constructs solutions iteratively. That is, each ACO agent constructs a trajectory by iteratively appending feasible maneuvers to the current solution. This allows for easy management of cluttered environments where obstacles are a concern. Once an obstacle is detected the set of feasible maneuvers is trimmed and construction continues. No backtrack search or repair method is required. This is in contrast to GA solution construction where the crossover operator is used to construct an entire trajectory in one step. This leads to a solution which is not guaranteed to be feasible and may need to be repaired. Formulating a good repair algorithm can be a difficult task. Is is possible that the trajectory created by the crossover algorithm can not be repaired and needs to be discarded entirely. This can lead to wasted computational effort which is detrimental to real-time performance.

1.3 Thesis Outline

The remainder of this thesis is organized as follows: Chapter 2 will give a review of previous solution formulations for this problem, along with a comparison of which ones have the potential to meet the algorithm requirements. The focus will be primarily on maneuver-based optimization approaches. More detailed discussion of the ACO and GA algorithms will also be given. Chapter 3 will give a detailed formulation of the algorithm used. Insight will be given into the particular design decisions. Chapter 4 will present the results from running the algorithm on a number of test cases. Analysis of expected versus actual results and comparisons between algorithms will also be made. Finally, Chapter 5 presents conclusions, final analysis, and some suggestions for future work.

Chapter 2

Background

2.1 Motivation

The benefit of using UAVs instead of manned vehicles in hostile situations is reduced risk to human pilots. Using UAVs in hostile environments poses some operational problems that must be handled. For example, loss of communications with a ground station may happen during a mission either due to terrain, long distance, or interference from enemy sources. Also, reduced operator workload is highly desirable. It currently takes several human operators to operate one UAV. Also, in the case of manned and unmanned vehicles flying together in a team, if a critical situation arises, the pilot's last concern is the health of the UAV. Thus the UAV must be able to autonomously generate a trajectory that will maximize survivability while possibly still continuing to satisfy the mission objective.

The need for greater autonomy in unmanned air vehicles (UAVs) is evident. With the existence of reliable control systems that are capable of following trajectories, even if those trajectories take the vehicle into its nonlinear flight regime, the next step is to go about generating those trajectories autonomously. This problem falls under the general moniker of motion planning. The general problem of robot motion planning has been studied for several decades. The quintessential book by Latombe [19] is an excellent place to get the fundamentals of the field. While the motion planning problem encompasses many different levels and complexities of specific problems, we will concern ourselves only with a specific instance: that of generating feasible trajectories for a UAV that will maximize the UAV's chance of survival in a hazardous environment.

With the idea in mind that a motion planning algorithm will eventually be placed on an autonomous vehicle there are several key attributes that a good algorithm should have:

1. Real-time performance

2. Ability to handle a non-linear objective function

3. Ability to handle large state spaces

4. Deterministic
5. Optimal

Of primary concern is the real-time aspect of the algorithm. Since threat locations will generally not be known ahead of time, the UAV must be able to react to pop-up threats by quickly generating an escape plan. Having a good solution in a matter of seconds rather than minutes is a necessity. Another key attribute is that the algorithm must be able to handle non-linear cost functions. For low altitude operations near ground-based threats, the use of terrain masking becomes an essential survival technique. This is the process of using the terrain to break the threat's line-of-sight (LOS) with the vehicle. The exposure function which we are minimizing will need to account for this terrain masking. Line-of-sight cannot be represented as a continuous parameter though. It is instead a binary function: either the vehicle is within the line of sight, or it is not. As such, the exposure function is non-linear. Also of relatively high importance is that the algorithm be able to handle state-space dimensionality of more than several states. The UAV motion planning problem takes place in a full 3D environment where position, velocity, time, and even possibly orientation of the vehicle play a role in characterizing a good solution to minimize exposure to the threat. Other attributes that might be desirable are if the algorithm is deterministic and/or optimal. Deterministic simply means that the same solution will be returned given the same conditions. Generally, deterministic systems are more predictable and possibly more reliable in the solutions produced, which may be important for flight validation. An optimal scheme is one which will return the solution corresponding to the global minimum/maximum of the objective function, if one exists. This might be asking for too much of the algorithm. For our purpose, getting a 'good' solution quickly is more important than getting the best solution possible if excessive computation time is required.

This thesis explores the application of the Ant Colony Optimization algorithm to the agile UAV motion planning problem. This chapter will give a brief overview of previous solution approaches for this problem. The primary focus will be on maneuver-based solution methods, though some time will be spent analyzing non-maneuver-based methods as well. These terms will be better defined in their respective sections. Finally, a detailed analysis of the Ant Colony Optimization heuristic will be developed. Some exposition on Genetic Algorithms will also be given since a currently existing GA motion planning solver will be used as the benchmark to test the performance of the implemented Ant Colony Optimization algorithm.

2.2 Previous Attempts

2.2.1 Maneuver-Based Motion Planning

The problem of motion planning for autonomous air vehicles is difficult for several reasons. Flying at low altitude in a hazardous environment requires that the aircraft be able to make full use of its aerodynamic capabilities. As such, many of the desirable trajectories will push the vehicle into a non-linear flight regime which is difficult to model precisely. The continuous nature of this non-linear flight envelope only increases the intractability of the problem. Using maneuvers as a basis set for trajectory generation is designed to alleviate these issues. In this thesis, a maneuver is used to generally refer to a smaller unit of a larger trajectory. Solving for an entire trajectory on board in real time can be a difficult task. Maneuver-based motion planning is the process of breaking down the trajectory into smaller pieces and solving for each of these maneuvers successively. The benefit is that it has the potential to greatly simplify the problem.

The idea of a maneuver automaton was formalized by Frazzoli [10]. It was essentially a known input sequence and corresponding state trajectory that would move the vehicle from one trim trajectory to another. A trim trajectory is simply a steady-state regime which the vehicle can remain in for any amount of time. The major feature of these maneuvers is that there exist group actions under which the system dynamics remain invariant. As such, applying one of these group actions to the state does not affect the feasibility of maneuvers which could be executed. An example of a group action is rotation about a vertical axis: executing a left turn is feasible regardless of the initial heading of the vehicle. Either through non-linear optimization methods or capturing human pilot inputs, a set of maneuvers can be generated offline. These maneuvers should span the full flight envelope of the vehicle. Additionally, they are guaranteed to be feasible and may be implemented online using a feedforward controller. Given this (finite) maneuver library, the problem of trajectory generation can be reduced to a discrete optimization problem. Trajectories can be generated by ‘stitching’ together compatible maneuvers and trim trajectories. The idea is that this formulation will simplify trajectory optimization while still retaining the agile capabilities of the vehicle. Frazzoli implemented a path planner based on these maneuver automata using a Rapidly-exploring Random Tree (RRT) search approach. Random states were generated and a local planner was then used to attempt to connect the random state to the nearest state in the tree. Using this scheme it was possible to generate trajectories through cluttered environments, with both moving and stationary obstacles.

The maneuver automata used by Frazzoli were rather rigid structures. Dever [7] expanded upon them by introducing a means of trajectory interpolation. With this it became possible to parameterize the maneuvers developed by Frazzoli, allowing for greater flexibility and efficiency in trajectory generation. Dever used a Mixed-Integer Linear Programming (MILP) framework to construct a path planner using these parameterized maneuvers. Schouwenaars [29] similarly used MILP to compute optimal trajectories for an agile helicopter. Schouwenaars also constructed a receding horizon version of the MILP planner. While MILP solvers are deterministic and optimal, their computational time grows exponentially with the size of the state space.

Not all maneuver-based planners have used the highly formalized maneuver primitives of Frazzoli. Indeed, maneuvers can often be reduced to simple heading changes over some fixed time interval. Dogan [8] used a probabilistic scheme to navigate through a 2D environment with known threat probability distribution. The ‘maneuvers' used were simple heading changes. The method used was essentially a one-step lookahead. In the regions where the exposure was less than some critical value, the vehicle would steer toward the goal point. Otherwise it would attempt to steer around regions of high exposure. The exposure map was generated by planting Gaussian kernels at various points in the 2D environment. As such, the regions maneuvered around were circular (and generally convex, when two or more circles didn't intersect) and a simple local search scheme such as this one was capable of finding usable trajectories. This scheme can run rather quickly, but handling hard constraints such as obstacles and terrain with this potential based method could lead to poor solution quality. By doing only one-step lookaheads the vehicle could quickly become stuck in a local minimum without enough space to maneuver around obstacles. Some form of backtrack search would then be needed to revise the plan.

Randomized techniques have proved useful for searching large state spaces where deterministic methods might get bogged down. Both LaValle [20] and Kim [18] have successfully used RRTs. The benefit is that they can easily handle non-holonomic constraints. The main difficulty lies in finding a way to uniformly search the state space, especially in the case where obstacles exist.

The motion planning problem can also be cast as a Markov Decision Process (MDP). An good source for general MDP knowledge is Russell and Norvig [27]. A classical MDP problem is movement of a robot in a 2D grid environment. The same framework can be applied to higher dimensional problems such as the one under investigation, but the solution methods remain the same. Generally some form of value or policy iteration is used to determine an optimal set of actions. The difficulty still remains that searching through a large state space can require large computational effort and some form of simulation or approximation is needed. Marbach [21], for example, uses a simulation scheme to update a parameterized policy. This need for simulation and learning of the cost-to-go function can negate the possibility of real-time application.

Genetic algorithms (GAs) have been applied to many different optimization problems in many different fields, and motion planning is no different. Ahuactzin [1] used GAs to solve for the trajectory of a robot in a 2D environment with obstacles. The maneuvers (chromosomes to use the GA terminology) used were simple heading changes. Crossover was done in the usual manner by encoding the heading changes as bit strings and then swapping substrings from two individuals. Sauter [28] solved a higher-level motion planning problem. He planned a path through a hexagonal 2D grid environment with threats. Vehicle dynamics were not explicitly accounted for. The specific problem under investigation of motion planning for a UAV in a threat-laden environment with terrain has been solved using a GA [24]. More detail is available in Section 2.4.

2.2.2 Non-Maneuver-Based Methods

In addition to the maneuver-based methods described in Section 2.2.1, numerous non-maneuver-based methods have been developed. These generally try to solve for the entire trajectory as a whole, rather than breaking it down into smaller pieces. A simple 2D planning case where the exposure is described by a density function (similar to that in [8]) was solved by Bortoff [5] using Voronoi cell decomposition. Voronoi cells were constructed based on the potential function and the edges were then used to construct a minimum exposure path. To account for dynamic constraints, the sharp corners of the cells were smoothed using a set of virtual forces that deformed the edges in a manner similar to a chain of masses and springs. Beard [3] also used Voronoi cells to solve for the problem of having multiple UAVs service multiple targets. Again, some post-processing is needed to form dynamically feasible trajectories. Asseo [2] solved a similar problem, but rather than using a density function to describe exposure, the threats were well-defined circular regions. The solution method generated a shortest path by traveling along tangents of the circular regions and then heading directly toward the goal when possible. In the first two cases, extension to higher-dimensional spaces and non-linear objective functions is difficult since formation of Voronoi cells in the presence of terrain and line-of-sight regions is ill-defined. In the third case, there is no natural way to account for trajectories which are forced to travel through the threat region and use terrain masking to avoid exposure.

Several more analytic optimization methods have also been used for motion planning problems. Toussaint [32] used H^∞ optimization techniques to tune spline parameters. An RRT search was superimposed to handle obstacle avoidance. This had the benefit of explicitly incorporating the vehicle dynamics, but no optimization is performed. The goal is simply to find *a* trajectory that reaches some final state, not the best trajectory to do so. Godbole [13] also used a spline-based representation to solve for entire trajectories. In that case, a wavelet-based optimization was used to transform the splines into wavelet coefficients and then tune those coefficients. The benefit is that it is a naturally multi-resolution scheme capable of handling different time scales. Optimization was either by an interior point method (for short term optimization) or evolutionary directed random search (for long term).

2.3 Ant Colony Optimization

Ant Colony Optimization (ACO) is often referred to as a metaheuristic [9]. That is, it is a very general method for solving combinatorial optimization problems. ACO is an agent-based evolutionary scheme in which each agent acts in a probabilistic manner to iteratively construct solutions. The way in which agents iteratively construct solutions fits well with a maneuver-based approach. As the name suggests, ACO is loosely based on the foraging and colony behavior of ants. Ants communicate with one another by depositing pheromone as they walk. This pheromone is detected by other ants which are then more likely to follow the same path as the previous ant, rather than wandering in some random direction. Using this relatively simple mechanism, it is possible for a colony of ants to find the shortest path to a food source. To see this, consider the case where there are only two paths to a food source from a colony, one short and the other long. At first there is no pheromone deposited on either path and ants will not prefer one path over the other. As time progresses and ants start carrying food back and forth to the colony the ants taking the shorter path will be able to make more round trips in a given amount of time than those taking the long path. As such, more pheromone will be deposited on the short path and more ants will begin to take that path. Similarly for the ACO algorithm, each agent constructs solutions based on 'pheromone' left by previous agents and then deposits its own pheromone based on the quality of the solution constructed. Pheromone deposit management is the most critical aspect of any ACO.

Using ant behavior as an optimization paradigm was originally outlined in Dorigo's Ph.D. thesis, and he has since written a book [9] detailing its origins and many applications. The original scheme was known as Ant System (AS) and it was applied to relatively small instances of the well-known travelling salesman problem (TSP). While the approach was novel, it was not competitive with modern TSP optimization schemes. Several adjustments were made and ACO was the result.

In addition to TSP, the ACO algorithm has been tested on other $\mathcal{NP}$-complete combinatorial problems. Gambardella [12] used a multiple-colony approach to solve the vehicle routing problems with time windows. The solution quality and computation time were competitive with existing methods. Gambardella [11] also combined ACO with a local search algorithm and solved instances of the sequential ordering problem. Again, the results were competitive with existing methods.

Several variants of the ACO algorithm have been developed in an attempt to increase performance overall or on a specific problem. The first such variant was the elitist ACO developed by Dorigo. In it, only the best ants from each generation are allowed to update the pheromone trails. The rank-based ACO [6] is a slight extension of this in which only the best n ants deposit pheromone, and the amount deposited is proportional to their rank. In an attempt to more explicitly control pheromone levels, Stützle [31] developed the $\mathcal{MAX} - \mathcal{MIN}$ ACO in which upper and lower limits are placed on the amount of pheromone permitted on any arc. ACO is best suited for discrete optimization, but attempts have been made to apply the principles to continuous domains [4, 30]. In these cases though, the test cases used were rather trivial problems.

Merkle [22] applied several ACO variants to the resource-constrained project scheduling problem. In most cases, ACO outperformed other heuristic approaches such as genetic algorithms and simulated annealing. Some attempts have even been made to merge the ACO and GA schemes. Karimifar [17] developed an ant conversation extension in which ants are capable of combining partial solutions to form new solutions that may not have been visited otherwise. As parameter tuning can be a difficult proposition when implementing a ACO algorithm, Rice [26] used a GA overlay to optimize the parameters used as the ant colony was evolving.

2.4 Genetic Algorithms

In the field of general heuristic methods, genetic algorithm [16] (GA) is certainly one of the more popular ones. It has been applied to numerous combinatorial and optimization problems. Goldberg's book [14] gives a good explanation of some of the fundamentals and potential applications for GAs. Like ACO, GA is an agent based method in which populations of agents act cooperatively to achieve some optimization goal. The main difference is the way in which the two share information between agents. The fundamental operator in GA is the crossover operator, which is meant to take two solutions in the current population and combine information from both to generate a new solution. Ideally this new solution will inherit the positive qualities of both of its parents and have a higher fitness. Finding a way to effectively implement the crossover method is the primary challenge in any GA implementation.

Pettit [24] used a GA to solve almost this exact problem instance. Pettit's paper will be the basis for the GA implemented in this thesis to be used as a comparison. In this case, the goal is to minimize exposure to stationary, ground-based threats. If the vehicle is within range of a threat and in its line-of-sight, the vehicle's exposure increases. The benefit of GA is that it can easily handle such a non-linear, discrete objective function.

It is necessary to find a proper solution encoding to use a GA. That is, a way to represent a solution to the problem as a series of genes capable of crossover. In Pettit's case, a solution consisted of a series of command inputs including a change in velocity magnitude (Δv), change in heading angle ($\Delta\psi$), and change in climb angle ($\Delta\gamma$). Each command input was assumed to last for one second. From this set of relative command inputs it was possible to decode it into a series of absolute positions, suitable for fitness evaluation.

The exposure (fitness) function used is a function of absolute position. Thus when two solutions are combined it is desirable to keep the absolute locations of part of each. The difficulty with this is that we then face the task of finding a way to transition from the end of one trajectory segment to the beginning of another. Pettit did this using a two-circle method in which two constant radius arcs are used to match velocity and position at the end of each segment. The problem is that this addition to the trajectory could possibly add an excessive amount of exposure and ruin the quality of the solution. Also, there is no guarantee that this repair method will even return a feasible solution, since the added segment might intersect the terrain.

2.5 Algorithm Comparison

Comparisons with existing methods, especially GA, are common. The reason for this is the similarity between ACO and GA. Several benefits of using either heuristic method are apparent. First, both are something of an anytime algorithm. That is, a feasible solution is available at any point after the first generation has completed. Letting the algorithm run for longer periods of time will simply improve solution quality, to a point. Second is the ability to handle non-linear cost functions. The agents do not need to know anything about the structure of the function they are trying to optimize. All that is required is that, given a full solution, the cost function may be evaluated to a real number. The real-time characteristics of using ACO for motion planning are not fully known; this is what this thesis intends to investigate. The runtimes for several TSP instances presented by Dorigo are promising, and it would seem that runtime performance can be adjusted based on formulation, but at the expense of solution quality. Similarly, it is not clear how well ACO can handle high-dimensional state-spaces. This will be another area of investigation for this thesis.

Several possible shortcomings of ACO are also evident. First, stagnation can be a problem. This is when the colony quickly gets stuck in a local minimum and is unable to improve solution quality as time progresses. This seems to largely be a function of how pheromone levels are handled. Another possible shortcoming is that ACO, like GA, is not a deterministic algorithm. This may be seen as a shortcoming by some, since different results may be given for the same problem. Yet non-deterministic algorithms can perform well in large optimization problems where more systematic searches may fail or take too long. Finally, ACO is not really an optimal algorithm. A convergence proof is given in [9] which states that the colony will eventually find an optimal solution, but no estimate on the time of convergence can be given. Thus, it may take infinitely long to find the best solution.

Table 2.1 presents a summary of the characteristics of several of the algorithms described so far. With these attributes in mind, ACO seems like a logical choice for a motion planning solver. The goal of this thesis is to determine whether or not ACO is a viable solution method for the problem of trajectory generation in a hazardous environment. A genetic algorithm will be used as a benchmark to gauge its performance. This comparison is natural considering the similarities between ACO and GA. As seen in the table, ACO and GA share similar algorithm characteristics. These similarities will be discussed further in Chapter 4.

Attribute	ACO	GA	MILP	Voronoi Cell	RRT
Real-time	Yes (See Section 4.3)	Yes (See Section 4.3)	No	Yes	No
Non-linear cost function	Yes	Yes	No	No	Yes
Large state space	Yes	Yes	Yes	No	Yes
Deterministic	No	No	Yes	Yes	No
Optimal	No	No	Yes	Yes	No

Table 2.1: Motion Planning Algorithm Characteristics

Chapter 3

Problem Formulation

3.1 Dynamic Constraints

The problem under consideration is the so-called UAV motion planning problem. Formally, the problem may be stated as:

$$\begin{aligned} \min \quad & J(\mathbf{x}(t), u(t)) \\ \text{s.t.} \quad & \dot{\mathbf{x}} = f(\mathbf{x}, u) \\ & \mathbf{x}(t) \notin \mathbb{T} \qquad t \in [0, T] \end{aligned} \tag{3.1}$$

Where $\mathbb{T}$ is used to represent terrain. Hence the constraint $\mathbf{x}(t) \notin \mathbb{T}$ simply means that the state trajectory does not intersect the terrain. The dynamic system under consideration is a UAV. Its dynamics are generally non-linear and can be described by a set of differential equations, $\dot{x} = f(x, u)$. The general motion planning problem is to select an input profile, $u(t)$, that will minimize some cost function, $J(x(t), u(t))$. While in some cases it may be possible to linearize the dynamics to simplify the problem, this is not always desirable. Often the non-linear dynamics region is where the vehicle is able to perform at its maximum capabilities and has the potential to lead to significantly better performance.

Working directly with the differential form of the dynamics is generally very difficult, and undesirable. The work on maneuver automatons done by Frazzoli [10] allows us to simplify the representation of this complex dynamical system. By limiting our actions to those of known maneuver primitives we transform this continuous optimization problem into a discrete one. That is, rather than considering all possible input profiles, $u(t)$, we will consider only those input segments which have been learned *a priori* and result in predictable behavior. Using a known dynamics model it is possible to generate this feasible maneuver set offline and store it in a maneuver library.

The dynamics model used for this thesis is the same one used by Pettit [24] for his GA construction. Maximum acceleration and deceleration are a function of velocity magnitude. Maximum and minimum velocity constraints are also imposed. A polynomial curve fit of Rate-of-Climb (ROC) as a function of velocity is used to determine the upper limit on flight path angle increase.

One additional constraint that is particularly important is obstacle avoidance. In this motion planning problem terrain is the main concern for obstacle avoidance. Clearly any trajectory that intersects the ground is infeasible. However, it is usually desirable to fly relatively close to the ground in hazardous environments and use the terrain to mask any threats. As such, we would like to be able to fully exploit the vehicle's ability to climb, dive, and bank to hug the terrain.

3.2 Maneuvers

A 'trajectory' in this thesis refers to a finite sequence of states, $\{x_i\}$, in some state space, $\mathcal{X}$, and an associated sequence of maneuvers , $\{M_i\}$, that describe the movement between these states. The maneuvers implemented here are similar to those described by Frazzoli[10]. The result of executing a maneuver is a change in state that can be described by a simple offset applied to each element in the state. Thus a maneuver can be represented as $M_i = \Delta x_i$. Given an initial state decoding a series of maneuvers is a simple matter of iteratively adding the offset from each maneuver:

$$x_{i+1} = x_i + M_i \quad i = 1 \dots N - 1 \tag{3.2}$$

The maneuvers are invariant under group action. That is, applying a symmetry transformation to the state does not affect the system dynamics (for a more precise definition refer to Section 2.2 of [10]). For example, consider a 2D environment where the state, $\mathbf{x} = (x, y, v, \psi)$, consists of position, velocity, and heading. The 'maneuver' of cruising 10 meters along the positive x-axis, $M = (\Delta x, \Delta y, \Delta v, \Delta\psi) = (10, 0, 0, 0)$, is known to be feasible. Then by applying the symmetry transformation of rotation about the z-axis we see that the maneuver $M = (\Delta x \cos\theta - \Delta y \sin\theta, \Delta x \sin\theta + \Delta y \cos\theta, \Delta v, \Delta\psi) = (10\cos\theta, 10\sin\theta, 0, 0)$ is also feasible for any θ.

$$M' = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta v \\ \Delta\psi \end{bmatrix} = \begin{bmatrix} \Delta x \cos\theta - \Delta y \sin\theta \\ \Delta x \sin\theta + \Delta y \cos\theta \\ \Delta v \\ \Delta\psi \end{bmatrix}$$

Thus it doesn't matter what heading the aircraft is flying. Similarly, translation is a group action, so it doesn't matter at what (x, y) location the aircraft begins. In this thesis these symmetry transformations are limited to translation and rotation about a vertical axis. For more detail read Section 3.4.1.

Before a maneuver can be executed, certain boundary conditions must be met. To state it formally, to execute maneuver M_i from state $\mathbf{x}_i$, $\mathbf{x}_i$ must belong to the set of initial condition states for the maneuver, $\mathbf{x}_i \in D(M_i) \subset \mathcal{X}$. Intuitively, each maneuver can be thought of as a function that maps states from some domain, $D(M_i) \subset \mathcal{X}$, to a range, $R(M_i) \subset \mathcal{X}$. Inclusion in a maneuver's domain is dependent on certain properties of the state. In this thesis this simplifies to matching velocity. That is, each maneuver has a specified initial velocity, and a state is included in the domain if and only if it matches that velocity. Using this representation, the motion planning problem then becomes:

$$\begin{aligned} \min \quad & J(\mathbf{x}_i) \\ \text{s.t.} \quad & \mathbf{x}_{i+1} = \mathbf{x}_i + M_i \qquad i = 0, 1, \ldots, N-1 \\ & \mathbf{x}_i \in D(M_i) \\ & \mathbf{x}_i \notin \mathbb{T} \end{aligned} \tag{3.3}$$

Four algorithms have been implemented for this thesis: 2D ACO, 3D ACO, 2D GA, 3D GA. In the 2D case the state consists of the tuple $(\vec{r}, v, \psi, t)$ where $\vec{r} \in \mathbb{R}^2$ is the position, v is the velocity magnitude, ψ is the heading, and t is the time. In the 3D case the state contains a 3D position vector $\vec{r} \in \mathbb{R}^3$ and also the flight path angle, γ. The same dynamic model was used in all four algorithms. Also, in each case every maneuver had the same fixed execution time, and the solution horizon was the same. The main difference between the ACO and GA is the types of maneuvers used.

In the case of the GA each gene that comprises a chromosome is a command $\{\Delta v, \Delta\psi, \Delta\gamma\}$, where Δv is a change in velocity, $\Delta\psi$ is a change in heading angle, and $\Delta\gamma$ is a change in flight path angle. Δv, $\Delta\psi$,and $\Delta\gamma$ can take on any value in the continuous feasible range defined by the dynamics model. These changes are assumed to occur at a constant rate during the maneuver. It should be noted that everything here is deterministic. There is no uncertainty when performing a maneuver as to what the final state will be.

The maneuvers used by the ACO are a discrete subset of those maneuvers used by the GA. Since ACO is a discrete optimization scheme, the maneuver library must adhere to this framework. Hence the Δv, $\Delta\psi$,and $\Delta\gamma$ used in ACO are only allowed to take on finitely many values in the feasible range. Thus, the capabilities of the ACO algorithm may appear to be limited compared to the GA. The idea though is that this restriction on trajectory construction will allow for faster computation times without a significant detriment to performance.

3.3 Trajectory Evaluation

Both ACO and GA are capable of handling non-linear cost functions. Indeed, this is one of their greatest advantages. The only requirement is that given a full trajectory, we are able to evaluate the cost function, J, to a real number. Several cost functions were implemented for this thesis. The simplest of these was distance to some goal point. This was implemented in both the 2D and 3D cases. More formally, given a sequence of N states and a goal state, $\mathbf{x}_f$, the function is:

$$J(\mathbf{x}) = min_{i=1\ldots N} |\mathbf{x}_i - \mathbf{x}_f| \tag{3.4}$$

Thus the goal is to come as close as possible to the desired state. Here the distance between states is simply the Euclidean norm, $|x_i - x_f| \equiv \sqrt{(x_i - x_f)^2 + (y_i - y_f)^2}$. Only position is specified; time, velocity, and heading are not accounted for. A similar function would be one which only accounts for the final state. That is:

$$J(x) = |\mathbf{x}_N - \mathbf{x}_f| \tag{3.5}$$

In some sense though this is a more difficult condition to minimize than Equation 3.4 since it requires that the time horizon be approximately right.

A more interesting cost function, and the one at the core of maneuvering in a hazardous environment, is that of exposure minimization. Each ground-based threat in a hazardous environment has a position, p, a effective lethality range, R, and a lethality weight, L. Given m land-based threats the exposure function to minimize is then:

$$J(x) = \textstyle\sum_{i=1}^{N} \sum_{j=1}^{m} E_j(\mathbf{x}_i) \tag{3.6}$$

Where $E_j : \mathcal{X} \to \mathbb{R}$ is the exposure function for the jth threat. The exposure function implemented in this case was:

$$E_j(\mathbf{x_i}) = \begin{cases} b(\mathbf{x_i}) L_i (1 - \frac{|\mathbf{x_i} - p_j|}{R_j}) & \text{if } |\mathbf{x}_i - p_j| < R \\ 0 & \text{if } |\mathbf{x}_i - p_j| \geq R \end{cases} \tag{3.7}$$

$$b(\mathbf{x}) = \begin{cases} 1 & \text{if within line-of-sight} \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

The $b(\mathbf{x})$ is a binary switch used to account for line-of-sight. That is, if the terrain occludes the vehicle from a target, it does not take any exposure from that target. Clearly this cost function is non-linear. The discrete nature of the line-of-sight check can make traditional continuous optimization schemes such as gradient descent difficult.

Other exposure functions are certainly possible. The exposure function used by Pettit [24] was something similar to a Gaussian distribution, while constant functions and functions that decay as $1/x^2$ have also been used. Any of these would work for evaluating the effectiveness of the ACO algorithm. The goal is to see if ACO can exploit LOS to reduce exposure.

3.4 ACO Implementation

At this point some more discussion of the problem representation is warranted. Recall that the trajectories constructed by ACO are sequences of maneuvers from a finite maneuver library. Thus the problem may be abstracted to one of choosing $N - 1$ branches in a decision tree. Each node of this decision tree corresponds to a state, and each branch corresponds to the feasible maneuvers that can be executed from that state. Associated with each branch are two values: a pheromone value, τ, and a heuristic value, η. It is these two values that will guide an agent during solution construction, and which will be modified based on information learned from previously constructed solutions. The time horizon of the problem is on the order of 20 seconds. Each maneuver lasts a fixed amount of time, usually one or two seconds.

```
defineManeuverLibrary();
initializePheromoneTree();
while Termination Conditions not met do
    for each Ant in colony do
        constructSolution();
        updatePheromones();
        daemonActions();
    end
end
```

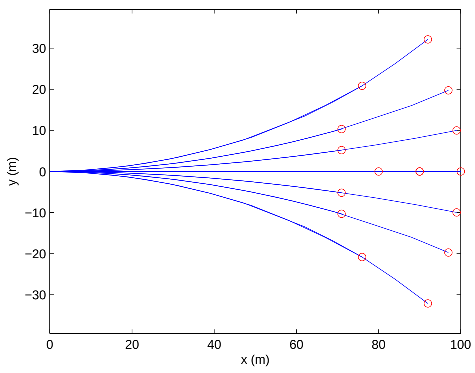
Algorithm 1: ACO Pseudocode [9]

ACO is an evolutionary, population-based optimization algorithm. Algorithm 1 shows the basic steps implemented for this ACO scheme. The primary operations needed are *constructSolution* and *updatePheromones*. As the name suggests, *constructSolution* is responsible for constructing a full trajectory using the current pheromone and heuristic information. It uses the maneuver library produced offline by *defineManeuverLibrary*. Once a solution has been constructed and evaluated *updatePheromones* is used to update the relevant pheromone values. The *daemonActions* function is any centralized operation that requires global information. In this implementation it will take the form of an elitist update strategy which allows only the top 25% agents in the population to deposit pheromone. The remainder of this section will give more detail into possible implementations for each method, and the reason behind the specific implementation chosen for this design.

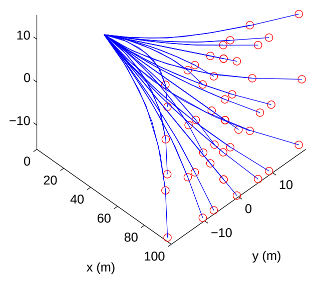
3.4.1 Trajectory Generation

A trajectory consists of a sequence of states. Transitions between states are described by maneuvers. Thus a full trajectory will consist of N states and $N - 1$ maneuvers. Note that this representation is used as a matter of convenience. For each maneuver the entire state trajectory as a function of time is known. A sampling of states is used to make evaluating the cost functions easier. It is essentially a discretization of an integral cost.

Before discussing trajectory generation, the idea of the maneuver library must first be explained. Recall that the ACO only used finitely many maneuvers. Also, these maneuvers can be rotated and translated freely without affecting feasibility, allowing us to stitch them together to make larger trajectories. The maneuver library stores this finite set of maneuver 'templates'. Refer to Figure 3-1 to see an example of such a library. Each maneuver is two seconds in duration. The straight cruise segments are parameterized by initial velocity, final velocity, and change in elevation. The turns are parameterized by velocity, turn radius, and elevation change.



(a) 2D Maneuver Library



(b) 3D Maneuver Library

Figure 3-1: Example Maneuver Libraries

Each agent (ant) of the ACO algorithm generates a feasible trajectory by stitching together maneuvers from the maneuver library. Each solution consists of a fixed number of maneuvers. Given an ant at state $\mathbf{x}_i$ it will choose which maneuver to execute next based on the pheromone and heuristic information available. Specifically, each feasible maneuver has associated with it a pheromone value, τ, and a heuristic value, η. The next maneuver is selected according to a pseudo-random distribution.

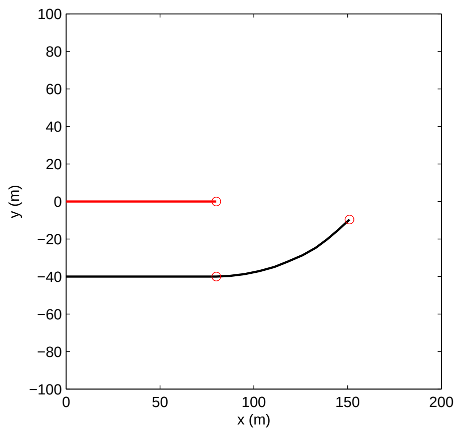
$$P(M_i) = \begin{cases} \frac{\tau_i^\alpha \eta_i^\beta}{\sum_i \tau_i^\alpha \eta_i^\beta} & \text{if } q > q_0 \\ 1 & \text{if } q < q_0 \text{ and } i = \arg\max \tau_i^\alpha \eta_i^\beta \\ 0 & \text{if } q < q_0 \text{ and } i \neq \arg\max \tau_i^\alpha \eta_i^\beta \end{cases} \tag{3.9}$$

Where $P(M_i)$ is the probability that M_i will be the next maneuver selected. The pseudorandom factor, $0 \leq q_0 \leq 1$, is a parameter and $0 \leq q \leq 1$ is a uniformly random value. That is, with probability q_0 the ant will select what appears to be the best maneuver based on the pheromone and heuristic information. Otherwise, the ant randomly selects a maneuver based on a weighted probability distribution.

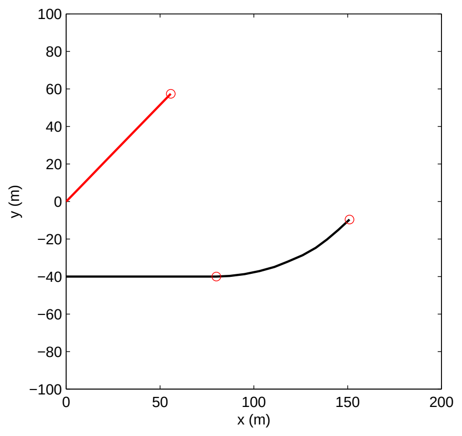
Once a maneuver has been selected the next step is to append it to the end of the current trajectory. It is important to note that the maneuvers in the maneuver library are simply templates. They must be rotated and/or translated to match the current terminal conditions. As an example, consider Figure 3-2. The maneuver template selected is a straight cruise. To patch this maneuver to the current trajectory we must first rotate, then translate the template.

3.4.2 Pheromone Update

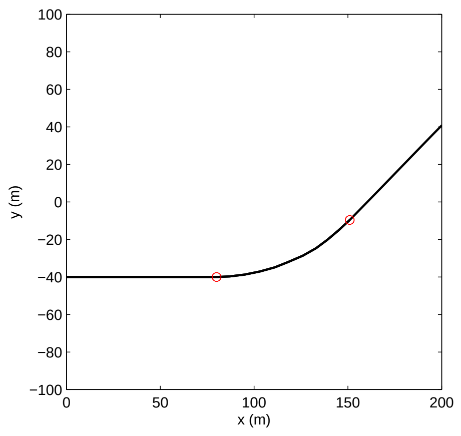
Pheromones are the primary means by which agents communicate. Once a population has constructed and evaluated solutions the next step is to update pheromones so



(a) Initial Maneuver



(b) After Rotation



(c) Translation to end point

Figure 3-2: Appending a new maneuver to an existing trajectory

the next generation can benefit from the information gained. Two pheromone update steps are executed every generation: pheromone evaporation and pheromone deposit. The problem of pheromone management is directly related to the population's ability to tradeoff exploration and exploitation.

Pheromone evaporation is the first step. The purpose of evaporation is to prevent stagnation by allowing the colony to 'forget' past solutions. That is, according to Equation 3.9 a reduction in τ_i will reduce the probability that maneuver M_i will be selected. There are several ways in which evaporation can be handled. The simplest approach is to reduce every pheromone by a constant factor, ρ, so that $\tau_{i+1} = \tau_i(1-\rho)$. Another possibility is to only remove pheromone along the paths which ants have travelled across. That is, in the decision tree which stores the pheromone values, only reduce the pheromone on a branch when that branch is selected by an agent. This is the method used by this thesis.

The purpose of pheromone deposit is to share information from one generation to the next and allow for exploitation of known good solutions. Several methods have already been developed for handling pheromone deposit. One method is to allow every ant to update based on a globally weighted score. A second is to allow every ant to update based only on local information. The elitist approach, developed by Dorigo [9], allows only the iteration-best and best-overall ants to update. The method used here was similar to the Rank-Based method proposed by Bullnheimer [6]. Only the best 25% of each generation and the best-so-far are allowed to update the pheromone trail. The amount deposited is proportional to the ants rank in the population. Since the goal is to minimize the objective function, $J(\mathbf{x_i})$, the pheromone deposited is proportional to $1/J(\mathbf{x_i})$. Saturation limits are placed on the amount of pheromone an agent can deposit at one time to prevent extreme bias towards good solutions. The saturation limit used was $10\tau_0$, where τ_0 is the initial pheromone value. This was done in an attempt to eliminate the need to scale the pheromone updates to each problem instance. An example of the pheromone deposit function is given by Figure 3-3.

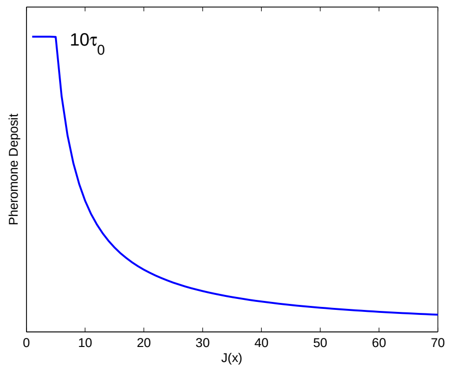


Figure 3-3: Pheromone Deposit Function

Initializing the pheromone value can be a difficult task. If the pheromone value is too low, the first several solutions will unduly bias the search, leading to stagnation. If the initial pheromone value is too high, the first several pheromone deposit cycles will be negligible and many generations will need to run before enough evaporation has taken place to lead to meaningful pheromone deposit. When used to solve the travelling salesman problem, the initial pheromone value used was $1/C^{nn}$, where C^{nn} is the nearest-neighbor heuristic solution. The idea was that this was relatively similar in magnitude to the pheromone deposits that would be made. One novel approach developed to handle this issue of pheromone levels was the $\mathcal{MAX} - \mathcal{MIN}$ ant system developed by Stützle [31]. In the $\mathcal{MAX} - \mathcal{MIN}$ formulation explicit upper and lower limits are placed on the pheromone levels.

3.4.3 Heuristic Information

One benefit of the ACO algorithm is the ability to explicitly incorporate problem-specific heuristic information. Notice that the probability distribution defined by Equation 3.9 includes both pheromone, τ, and heuristic, η, information. A good heuristic can be used to guide the algorithm toward promising solutions, leading to overall better performance.

In this thesis, heuristic calculation was performed using a one step lookahead. In the case where the cost function is given by Equation 3.4 the heuristic is the reciprocal of the distance to the goal point if the maneuver under investigation were executed.

$$\eta_i = \frac{1}{|(\mathbf{x}_{curr} + M_i) - \mathbf{x}_f|} \tag{3.10}$$

In the case where the cost function is the threat exposure given by Equation 3.6 the heuristic value is the reciprocal of the exposure incurred along the maneuver. In the case that the exposure is very small, so that its reciprocal is very large, a saturation value of $10\tau_0$ was used as the maximum allowable value. This is essentially the same scheme used for pheromone updates, Figure 3-3.

$$\eta_i = \frac{1}{\sum E_j(\mathbf{x}_{curr} + M_i)} \tag{3.11}$$

During the one-step look ahead, if it is discovered that the maneuver is infeasible due to collision both the pheromone and heuristic values are set to zero so that subsequent ants will ignore these options. Intuitively, that branch of the decision tree is pruned. It should be noted that these heuristic calculations are only performed once for each node.

3.4.4 Parameter Selection

When implementing an Ant Colony algorithm, several design parameters need to be selected, namely:

1. α pheromone exponent, Equation 3.9
2. β heuristic exponent, Equation 3.9
3. τ_0 initial pheromone value
4. q_0 pseudo-random factor, Equation 3.9
5. ρ evaporation factor

In order to minimize the amount of tuning needed, only two parameters(α and β) were adjusted for each scenario, while the remaining parameters were fixed. The initial pheromone value, τ_0, was set to 10. The goal of the implemented pheromone update rules was to remove scaling issues from pheromone maintenance. That is, usually the pheromone and heuristic values at each branch needed to be properly scaled to avoid one factor significantly outweighing the other. Doing this for every scenario would be tedious. Instead, a constant value of τ_0 was selected, and then all pheromone and heuristic updates were expressed as a multiple of τ_0. The pseudo-random factor, q_0, was set to 0.5. This value was selected after reviewing the results presented by Dorigo [9]. For large values of q_0 the algorithm would focus primarily on exploiting known good solutions, while for small values it would favor exploration of the search space. The evaporation factor, ρ, was set to 0.1 (again based on Dorigo's results). The larger ρ is, the quicker the algorithm is likely to 'forget' previously constructed solutions and explore new areas of the search space. The exponents, α and β, weight how strongly the decision of the next maneuver is influenced by pheromone and heuristic values, respectively. For large values of α one would expect the algorithm to quickly stagnate toward the first good solution found. Similarly, for large values of β the agents will be heavily drawn toward solutions with high heuristic value. Historical data were used to select reasonable value ranges for α and β. For each scenario several parameter settings were used to determine which produced the best results. The possible selections were $\alpha = \{1, 2\}$ and $\beta = \{2, 3, 4\}$. More discussion of these parameter sweeps is available in Section 4.2. Some factors are common to both ACO and GA. The population size was set to 8. The length of each maneuver was 2 seconds. The number of generations to execute before termination was set to 50. The correspondence between number of generations and runtime will be discussed further in Section 4.3. In short, a relatively small number of generations was selected to ensure small runtimes and the potential for real-time application.

3.4.5 Termination Conditions

The final design decision is how and when to terminate the algorithm. Left to run indefinitely the quality of the best-so-far solution will continue to increase to a point. One possible concern that must be addressed is stagnation. It is possible that the algorithm will quickly find a local minimum in the cost function and then heavily saturate the pheromone trail so that finding new solutions becomes increasingly unlikely. One possible way around this is to restart the algorithm once no improvement has been shown for a set number of generations.

One major benefit of ACO is that, by using the above implementation, every agent constructs a feasible solution every generation. Thus no matter when the algorithm is stopped a feasible solution is available (as long as one generation has finished of course). The scheme used here was to simply run the algorithm for a fixed number of generations, in this case 50.

3.5 GA Implementation

Like ACO, GA is a population based evolutionary algorithm. Each generation attempts to collect information about the solution space and then use this information to make future generations better. For a fully detailed explanation of GAs refer to Goldberg [14]. The major steps involved in implementing a GA are population initialization, selection, crossover(recombination), and mutation. Each of these will be explained in detail in the following sections. It should be noted that the implementation used here is very similar to that used by Pettit [24], since Pettit's thesis was used as a guideline.

To define some GA terminology, a chromosome is an entire trajectory. A chromosome is made of smaller building blocks known as genes. In this case each gene is a command of the form $(\Delta v, \Delta\psi, \Delta\gamma)$, where Δv is the change in velocity, $\Delta\psi$ is the change in heading angle, and $\Delta\gamma$ is the change in flight path angle. Each of these values is restricted to lie between an upper and lower bound which are determined using the dynamics model described in Section 3.1 based on the vehicle state. Since each gene is just a command and not a position, the chromosome must be decoded into a state trajectory before a cost function (fitness) evaluation can be performed. This is easily done by propagating forward in time from some known initial state. Each command has a fixed execution length and the prescribed changes are assumed to happen at a constant rate.

3.5.1 Population Initialization

Population initialization in the GA is a relatively simple task. Given an initial state, $\mathbf{x}_0$, command limits are computed as a function of this state: Δv_{max}, Δv_{min}, $\Delta\psi_{max}$, $\Delta\psi_{min}$, $\Delta\gamma_{max}$, $\Delta\gamma_{min}$. A command is then constructed by selecting values in these ranges according to a uniform random distribution. This command is then used to propagate to the next state where the process repeats until enough commands have been generated.

3.5.2 Selection

Selection is the process of choosing which members of the population are carried into the next generation or are recombined to form new members. The selection scheme used here was a simple roulette selection, where the probability that a member is selected is directly proportional to its fitness. One exception to this is the use of an elitist strategy where the best member from each generation is automatically carried over to the next.

3.5.3 Crossover

Crossover is the most critical operator in a genetic algorithm. It is the means by which beneficial information from one generation is carried over into the next. After being selected using the roulette method described above, two individuals are recombined with probability P_{cross}. During the crossover operation, two individuals are 'combined' to form a new one. This method of combining solutions is largely dependent on the representation used and the programmer's preference. It can also greatly affect the performance of the GA.

The crossover mechanism used here is the same used by Pettit [24]. Since the trajectory evaluation is based on the state trajectory, and not explicitly on the maneuvers or control inputs used, it makes sense to preserve the states from individual solutions when recombining them. The basic idea is to take the beginning section of one trajectory, and the tail section of another trajectory, and patch them together in some appropriate manner. The way in which trajectory segments are patched together is the so-called 2-circle method. Two tangent, constant-radius circles are constructed to connect the end state of the first segment with the initial state of the second segment. Figure 3-4 illustrates a 2D example of patching two trajectories together.

It should be noted that all the trajectories are constrained to be of a fixed length. As such, the solution produced by the crossover may need to be trimmed. This trimming can at times mean that no part of the second parent is included in the child. There are several potential problems using this recombination method. First,

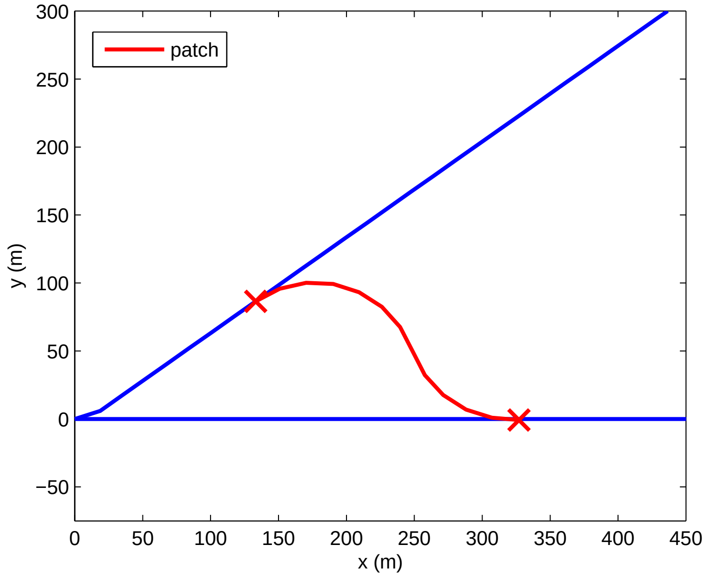


Figure 3-4: Trajectory crossover patching example.

the crossover is not guaranteed to be feasible. Either due to dynamic constraints or ground collision, the operation may fail. In the case that the crossover fails, it is attempted two more times using different crossover points. If both of these additional attempts fail the two chromosomes are moved into the next generation without generating a child. Also, adding this extra patch in the middle has the potential to ruin the solution quality. Generally, it is desirable to crossover two ‘good’ solutions and obtain a solution that is at least as good if not better. If the patch segment runs through an area of high exposure, the solution quality may be very poor. It would seem that physical trajectories are not naturally suited to crossover.

3.5.4 Mutation

The final GA operator needed is mutation. After a population is constructed each gene in each chromosome is mutated with some (small) probability, P_{mutate}. The mutation of a command involves choosing one the of components, (Δv,$\Delta \psi$,$\Delta \gamma$), and selecting a new random value from the feasible range. This mutation then of course requires that all states downstream of this point be recomputed. Also, there is the possibility that the mutation may result in an infeasible trajectory due to collision with the terrain. If this is the case, the mutation is discarded and the algorithm continues.

3.5.5 Parameter Selection

The two main parameters which must be specified are P_{cross} and P_{mutate}. P_{cross} is the probability that when an individual is selected it will be crossed over with another solution rather than simply transferred to the next generation. P_{mutate} is the probability that an individual will undergo mutation before being transferred to the next generation. Similar to what was done for α and β in the ACO case, multiple runs of the GA were conducted for several values of P_{cross} and P_{mutate}. The ranges used were $P_{cross} = \{0.5, 0.6, 0.7\}$ and $P_{mutate} = \{0.05, 0.1\}$.

3.5.6 Termination Conditions

Stagnation is still a concern with genetic algorithms, though this can be mitigated by selecting high values of P_{mutate}. Also, the algorithm can be restarted by generating a new random population when solution quality has not improved in a given number of generations. For this thesis the GA, like the ACO, was run for a fixed number of generations. In this case, 50 generations was used as the termination value.

Chapter 4

Simulation Results

This chapter will present the results obtained from running both the ACO and GA algorithms on a set of test cases. Several degenerate cases of the ACO algorithm were also used as baselines. Namely, since ACO utilizes both pheromone and heuristic information, tests were conducted in which *only* pheromone or *only* heuristic data were used. These correspond to the parameter selection of $\beta = 0$ and $\alpha = 0$, respectively, in Equation 3.9. Also, a purely random search was performed. This corresponds to the case where $\beta = 0$, and no pheromone updates are made by any agent. As such, there is a uniform probability distribution for selecting any maneuver at any given time. It should be noted that each of these three degenerate cases use the same maneuver basis set for constructing trajectories as ACO. Only GA uses a continuous set of available maneuvers. Again, the hypothesis is that even though ACO uses a more limited command set it will still be able to generate trajectories of superior quality to the GA.

At this point it is worth mentioning why a GA is being used as a comparison for ACO. The ACO and GA algorithms are remarkably similar. They are both population-based randomized evolutionary search schemes. As such, parameters such as population size and generation have the same meaning in both interpretations. Also, they have been formulated in such a way that the solution representation is nearly identical in each. Both use maneuvers of fixed length to construct full trajectories. The fact that both algorithms share such a similar structure makes comparison sensible.

Two metrics used to compare algorithms are best overall score per iteration and population average per iteration. Best overall score is the score (according to whatever cost function is being used) of the best solution found so far. Clearly this is a monotonically decreasing quantity. This metric should give some insight into how quickly the algorithm is able to converge on a good solution, and also whether or not the algorithm is prone to stagnation. The population average is the average score of every individual during a given generation. This gives an idea of how well the population as a whole is converging on a favorable solution region, as well as the relative emphasis given to exploration versus exploitation.

	Environment	threats/obstacles	objective
Scenario 1	2D	none	get to goal
Scenario 2	2D	2 obstacles	get to goal
Scenario 3	2D	8 obstacles	get to goal
Scenario 4	3D	terrain	get to goal
Scenario 5	3D	1 threat, terrain	minimize exposure
Scenario 6	3D	2 threats, terrain	minimize exposure
Scenario 7	3D	1 threat, terrain	minimize exposure and get to goal
Scenario 8	3D	2 threats, terrain	minimize exposure and get to goal

Table 4.1: Scenario Parameters

4.1 Scenario Results and Discussion

The test cases implemented were designed to start simple, and then progress to more difficult situations. The basic parameters of each test case are as follows:

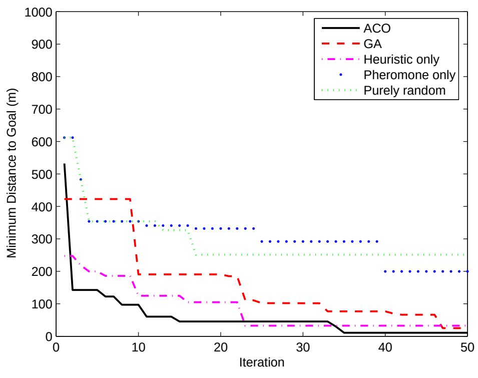
In the 2D case the obstacles are hard constraints. They are circular regions which the vehicle cannot enter. In the 3D case, the only hard constraint is the terrain. The threats are treated as soft constraints. That is, the vehicle may enter the threat region but there is a cost function penalty for doing so (Equation 3.7). For each case, the trajectory computed by each algorithm will be shown, as well as average population fitness and best individual fitness as a function of generation number.

4.1.1 Scenario 1

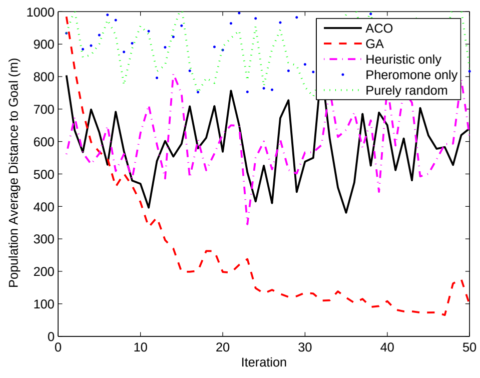
Scenario 1 is perhaps the simplest case imaginable. The objective is simply to get as close as possible to the goal point of $(x, y) = (900, 900)$ in a 2D obstacle-free environment. The vehicle begins at $(x, y) = (0, 0)$ traveling at 50 kts along the positive x axis. The score (fitness) is calculated according to Equation 3.4. Certainly there are better ways to solve this optimization problem. A simple A^* search, for example, could find the best set of maneuvers relatively quickly. The objective though is to see if these randomized methods are still capable of coming up with reasonable solutions. The results are presented in Figures 4-1(a)–(c).

As can be seen in Figure 4-1(c) all 5 methods produce reasonable results. Though the *Purely Random* and *Pheromone Only* approaches fail to get as close to the goal as other trajectories, the trend is for them to head in the right direction. This is a common trend that will repeat itself throughout these results: the *Purely Random* and *Pheromone Only* methods will perform poorly compared to the others.

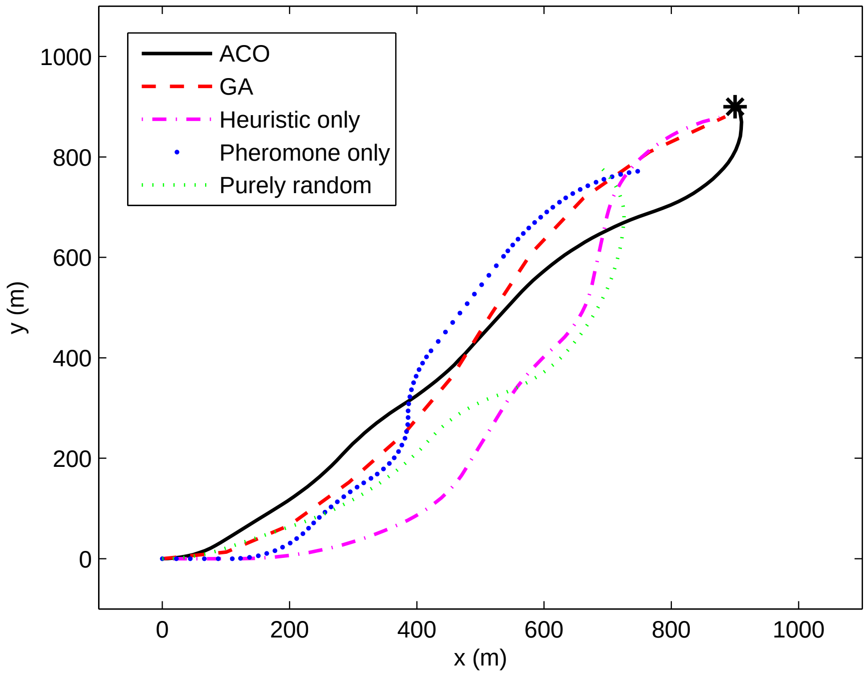
From Figure 4-1(a) we see that the ACO algorithm does indeed outperform the other methods. It not only converges faster (in a per iteration sense) but it also finds a trajectory closer to the goal than any other algorithm. One thing to note on this graph is the rapid improvement of ACO during the initial generations followed by much shallower improvement. This is possibly due to stagnation in the algorithm, which will be discussed further in Chapter 5. In contrast, the GA shows gradual, steady improvement throughout. This would imply that the GA is converging on a



(a) Best Overall Score

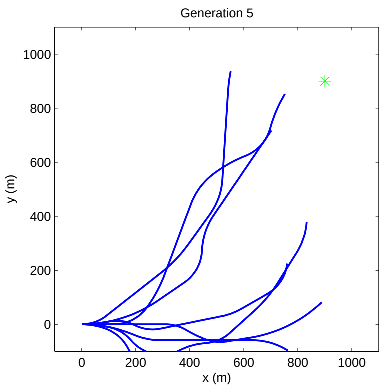


(b) Population Average Score

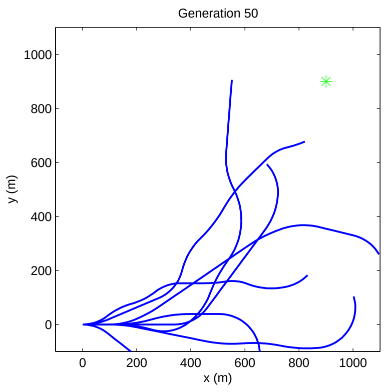


(c) Trajectories

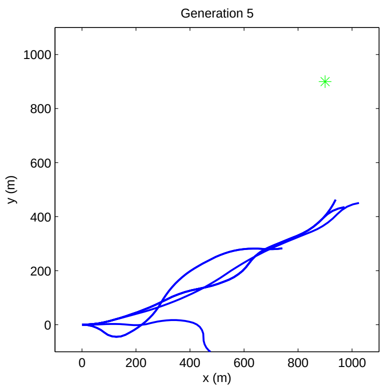
Figure 4-1: Scenario 1 Results



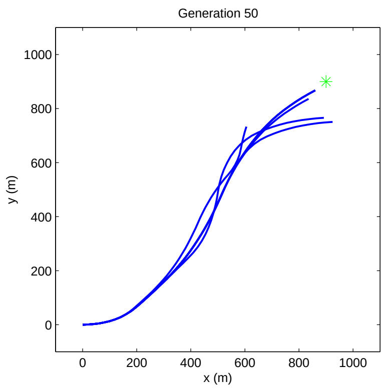
(a) ACO Generation 5



(b) ACO Generation 50



(c) GA Generation 5



(d) GA Generation 50

Figure 4-2: Scenario 1 Generation Comparisons

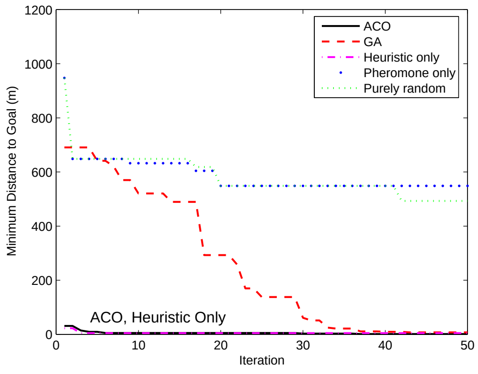
narrow solution region and heavily exploiting knowledge from previous generations. This is another trend that will appear throughout the results: rapid convergence and leveling-off of ACO solutions, and gradual improvement by the GA. This can be seen as a positive or negative aspect of the ACO algorithm and will be discussed further in terms of runtime impact and stagnation in Chapter 5.

Figure 4-1(b) shows the average score of the entire population at each generation. It is meant to give some insight into how the population is getting better by learning from previous generations. Also, it can lend some insight into the degree to which the algorithm is emphasizing exploration versus exploitation, and vice versa. It is evident that the GA population generally gets better on average from one iteration to the next. In contrast, the ACO average continues to fluctuate. This would seem to imply that a great deal of exploration of the search-space continues even during the later iterations. There are several possible explanations for this. First, the value of α used in Equation 3.9 may be too low. That is, since α controls how heavily pheromone value influences maneuver selection a low α would mean that pheromone information from previous generations is not contributing much to trajectory generation. The values used for this scenario were $\alpha = 1$ and $\beta = 3$. A second possible reason for this behavior is that not enough pheromone is being deposited by the top 25% agents to significantly affect subsequent generations. Indeed, as the number of available maneuvers grows, the effect of one agent depositing pheromone for a single maneuver is reduced.

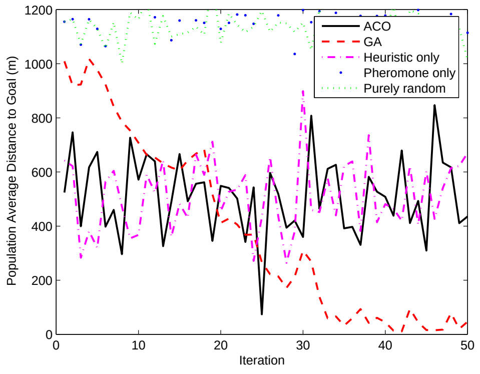
To better visualize this idea of population average fitness and convergence, consider Figure 4-2. Figures 4-2(a) and 4-2(b) show the entire population for the ACO algorithm at Generations 5 and 50, respectively. Clearly, there is still a great deal of exploration in both cases. In contrast, the GA population(4-2(c)–(d)) very tightly converges to a narrow solution space by the later generations. Notice that the best ACO trajectory from Figure 4-1(c) is not present in the last generation, Figure 4-2(b). Even though ACO uses an elitist strategy by heavily weighting the best trajectory found, it does not explicitly carry that solution through subsequent generations.

4.1.2 Scenario 2

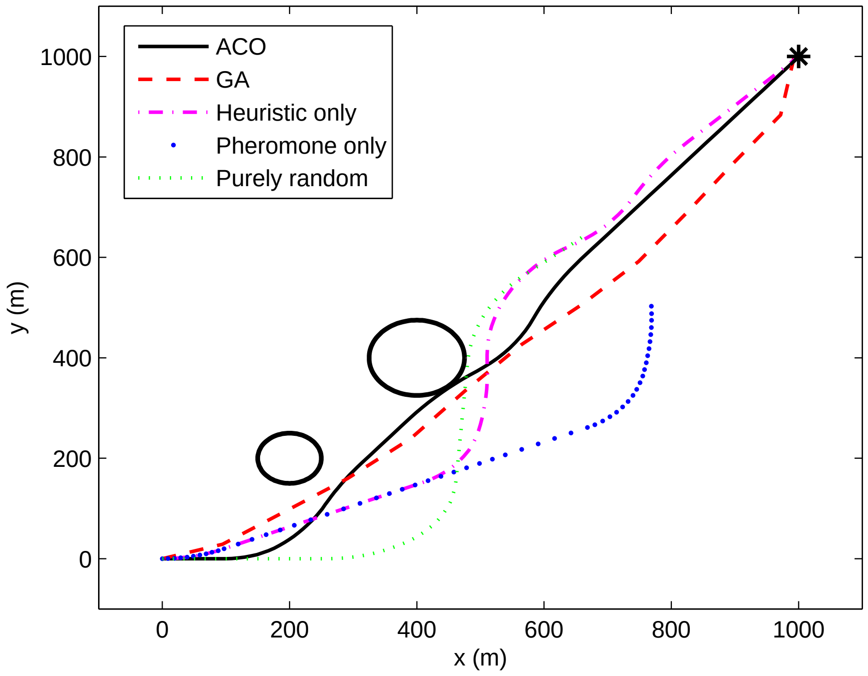
Scenario 2 is a slight modification of Scenario 1 in that a couple of obstacles have been added to make maneuvering to the goal more difficult. The vehicle begins at $(x, y) = (0, 0)$ traveling at 50 kts along the positive x axis. The score (fitness) is calculated according to Equation 3.4, with the goal point at $(x, y) = (900, 900)$. These obstacles represent hard constraints, in that any trajectory that intersects them is discarded as being infeasible rather than heavily penalized. Figures 4-3(a)–(c) show the best overall score, population average score, and trajectories computed. Nearly identical trends are seen as in scenario 1. ACO is quick to converge while GA is slower, but both find very good solutions in the end. It would appear that having a couple trivially placed obstacles does little to affect algorithm performance. In Figure 4-3(a) ACO and *Heuristic Only* are very close to one another. This is indicative of a strong heuristic influence on ACO performance. This will become a recurring theme as ACO closely mimics the decisions of whatever heuristic is used to guide it. Also, it would



(a) Best Overall Score



(b) Population Average Score



(c) Trajectories

Figure 4-3: Scenario 2 Results

appear that the heuristic was able to outperform the GA. This is not always the case and is dependent on what heuristic is selected. It is possible to incorporate heuristic information into GAs by using it to initialize the population, rather than using a purely random scheme. This would presumably improve GA performance, but was not implemented for this thesis.

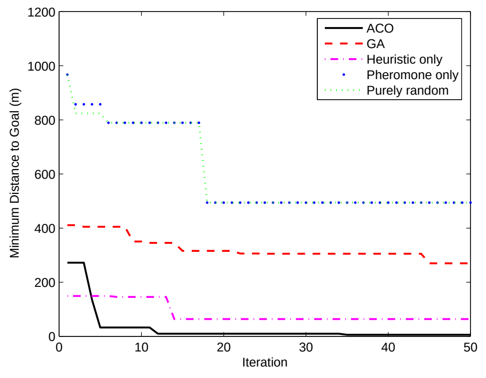
4.1.3 Scenario 3

Scenario 3 is a cluttered environment with many obstacles. This time, obstacle avoidance becomes a significant contributing factor to algorithm performance. The vehicle begins at $(x, y) = (0, 0)$ traveling at 50 kts along the positive x axis. The score (fitness) is calculated according to Equation 3.4, with the goal point at $(x, y) = (900, 900)$. Figures 4-4(a)–(c) show the results. GA performance has significantly decreased. This is most likely due to the limitations of the crossover operator. As mentioned earlier, crossover is the primary means by which GAs pass useful information from one generation to the next. With the presence of obstacles though, it is far more likely that the resulting child solution will intersect one of these obstacles and be infeasible. This is a significant detriment to GA performance. In contrast, ACO builds each trajectory iteratively allowing it to maneuver around obstacles rather than simply discarding solutions completely. This fundamental difference in solution construction and information propagation between ACO and GA is the core reason why ACO has the potential to out-perform GA for motion planning problems, especially in cluttered environments.

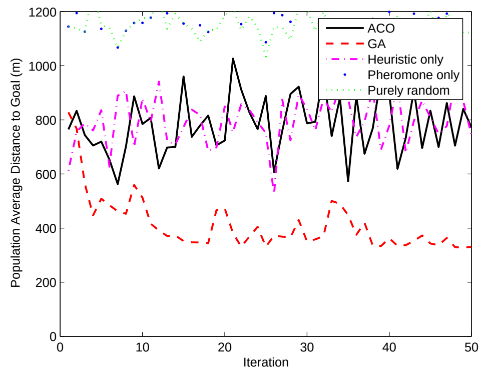
In both Figures 4-4(a) and 4-4(c) the *Pheromone Only* and *Purely Random* solutions lie directly on top of one another. This indicates that the amount of pheromone being deposited is small compared to the amount of pheromone already present on each branch. As such there is a nearly uniform chance of selecting any maneuver. To make this clearer, consider the case where there are 100 maneuvers available, each with an initial pheromone value of 10. The probability of selecting any single maneuver based on pheromone alone (Equation 3.9 with $\beta = 0$, $q_0 = 0$) is 1%. If an additional 10 pheromone is deposited on a single maneuver, the probability of selecting that maneuver is only raised to 1.9%, and the selection is still nearly uniform. This phenomena of pheromone influence decreasing as the number of maneuvers increases is a potential drawback to ACO and is discussed further in Section 5.1.

4.1.4 Scenario 4

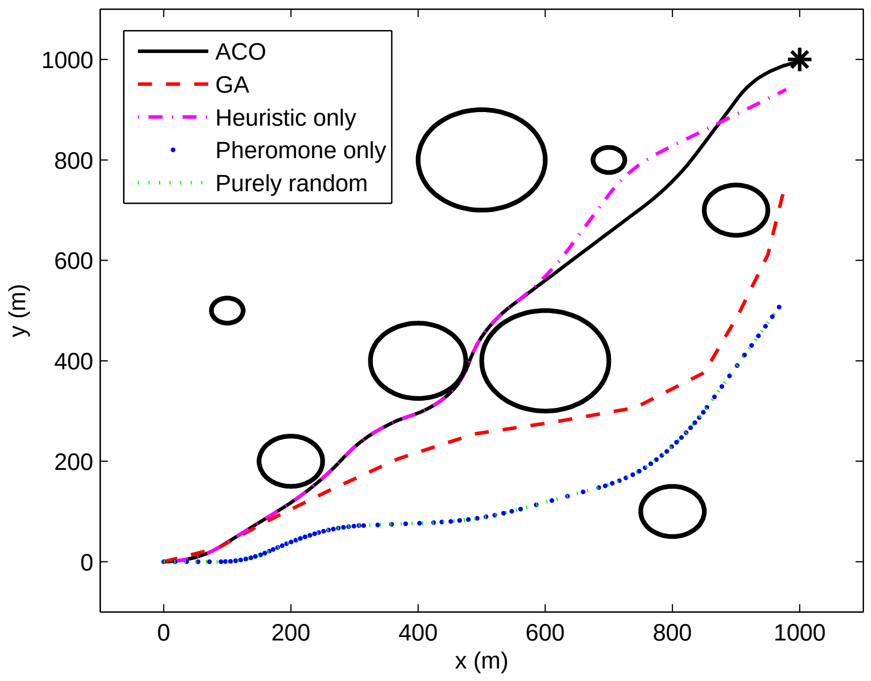
Beginning at scenario 4, the test cases involve a full 3D environment with terrain. The vehicle must now climb and dive in order to avoid collision with the ground. Recall that terrain collision is treated as a hard constraint. That is, there is nothing in the objective function to encourage the vehicle to fly low or high. In later scenarios where threats are present this will become an important issue, but for this simple case it has little influence. The vehicle begins at $(x, y) = (600, 550)$ at 100 m AGL, heading north (positive y axis) with a velocity of 50 kts. The maneuver library has now expanded to a total of 144 maneuvers. The goal point is at $(1000, 1600)$. The usual



(a) Best Overall Score

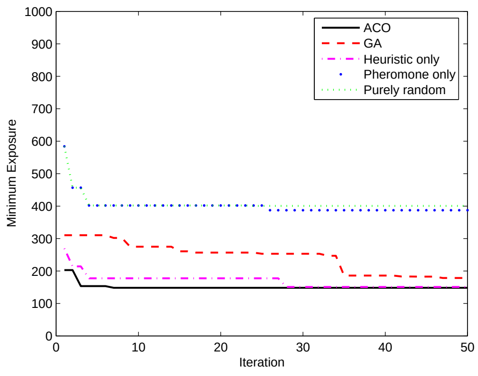


(b) Population Average Score

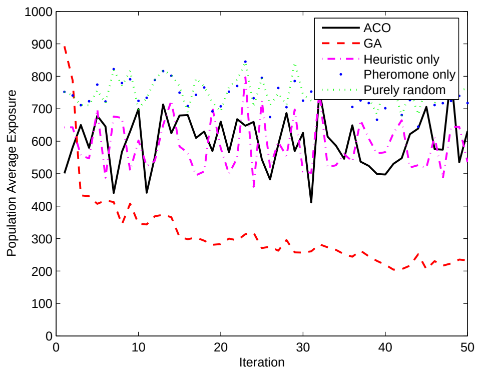


(c) Trajectories

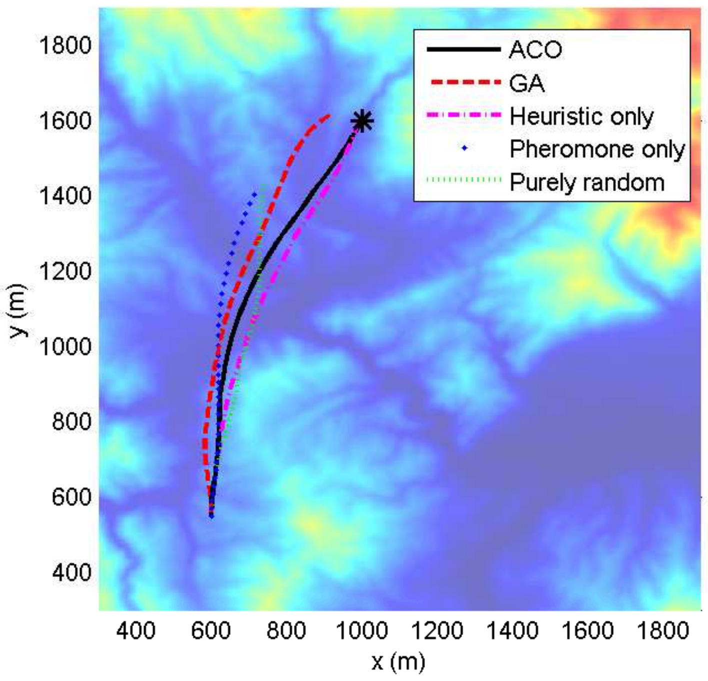
Figure 4-4: Scenario 3 Results



(a) Best Overall Score

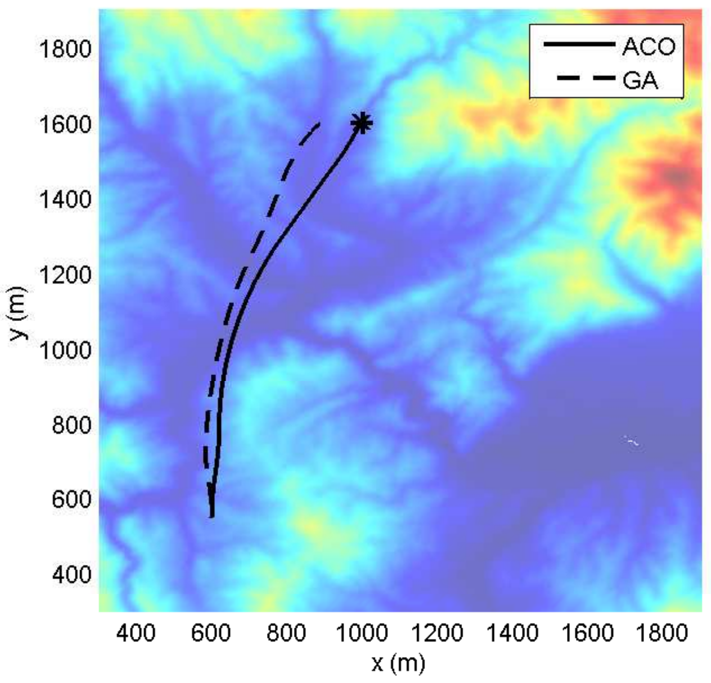


(b) Population Average Score

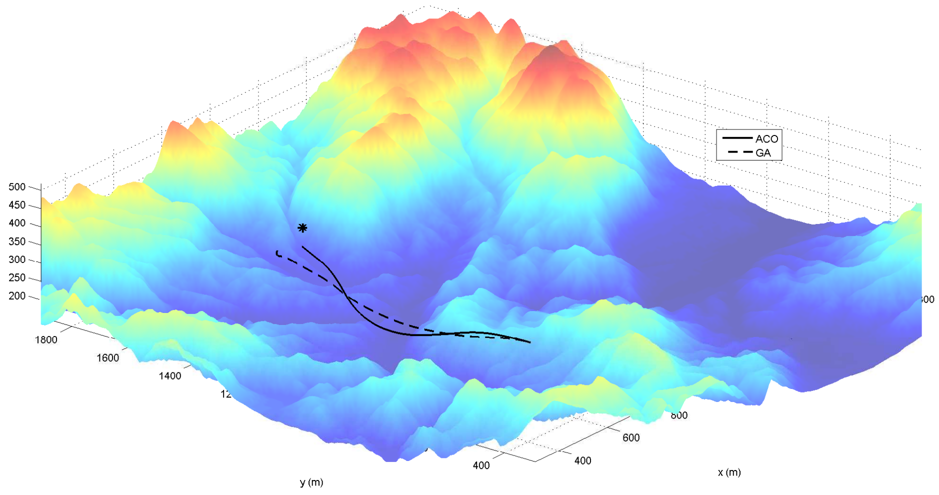


(c) Trajectories

Figure 4-5: Scenario 4 Results



(a) Orthogonal View



(b) Isometric View

Figure 4-6: Scenario 4 Trajectories

metrics are presented in Figures 4-5(a)–(c). The trajectories exhibit the expected behavior, with both the *Purely Random* and *Pheromone Only* cases performing poorly, and the *Heuristic Only* performing well. In Figure 4-6 only the GA and ACO results are shown for clarity. Figure 4-6(b) presents an isometric view of the ACO and GA trajectories. From this angle is is possible to see the elevation changes that both trajectories exhibit in order to compensate for the terrain.

4.1.5 Scenario 5

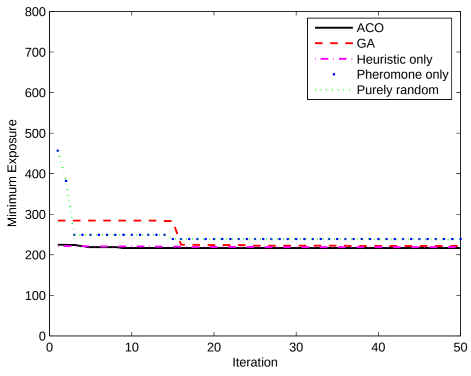
With scenario 5 the real problem of agile UAV motion planning begins. Here the vehicle starts at the same location, $(x, y) = (600, 550)$, as in scenario 4, but rather than trying to get to a goal point, its objective is to minimize exposure to a threat (Equation 3.6). A stationary ground threat is located at $(500, 600)$ with a threat radius of 700 m. A constant lethality, L_i (Equation 3.7), of 100 was used for this and all subsequent threats. Notice that the UAV begins in the threat envelope. This is designed to represent the case of a pop-up threat where the location of the threat is not known until the vehicle has already entered the threat envelope. The vehicle must then do its best to maneuver out of the threat region and minimize exposure. The results are presented in Figures 4-7(a)–(c). The black star is the threat and the associated circle is its exposure radius. Notably, the same trends present themselves as in previous cases. Again the *Purely Random* and *Pheromone Only* solutions are nearly identical. Both the ACO and GA methods find good solutions. Intuitively, these solutions make sense since the vehicle is simply turning away from the threat and flying away radially.

Recall that one of the primary means of reducing exposure is using the terrain to break line-of-sight. Figure 4-8 gives some insight into this phenomena. The ACO trajectory is reproduced from Figure 4-7(c). The points plotted in red are those that are within the LOS of the threat, while those in black are not within sight. Notice that before the vehicle leaves the threat radius it is able to break the line of sight. It does this by diving quickly after passing over a ridge. This behavior is easier to see in Figure 4-9.

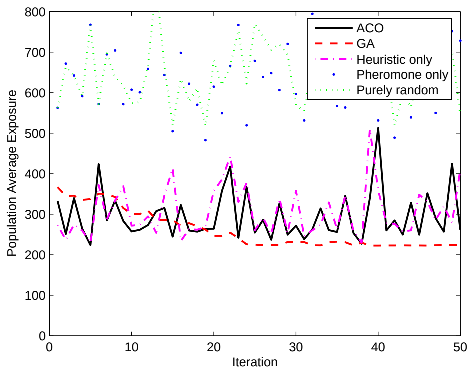
4.1.6 Scenario 6

Scenario 6 expands upon the previous scenario by adding an additional threat at $(1200, 800)$ with a radius of 500 m. Again, the objective is to minimize exposure. The vehicle begins at $(x, y) = (600, 550)$ at 100 m AGL, heading north (positive y axis) with a velocity of 50 kts. Here the GA and ACO algorithms produce very different solutions, Figure 4-10(c). The ACO takes the strategy turning away from the first threat as in scenario 5, and then turning away again when it is within radius of the new threat. The GA, on the other hand, decides to bypass the second threat completely and fly closer to the first threat.

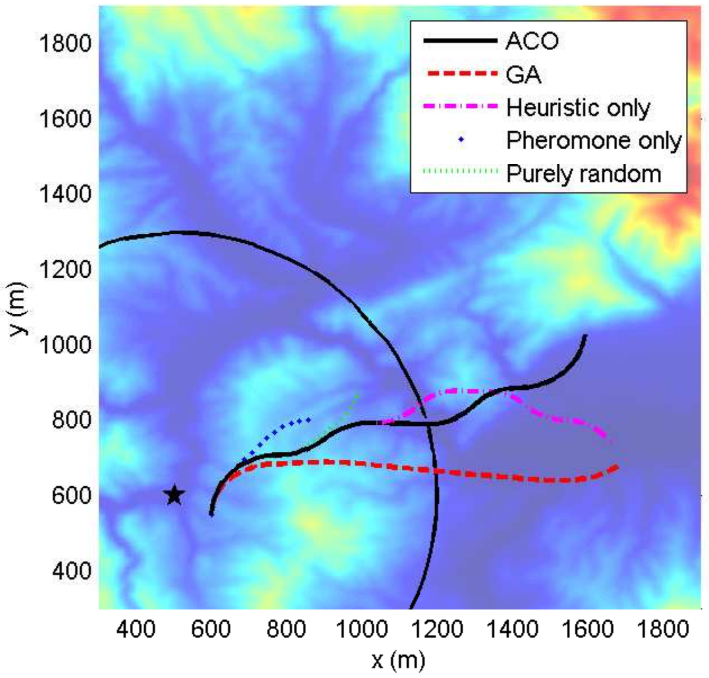
From Figure 4-10(a) it is evident that the ACO solution has the lowest exposure. To see why the ACO solution has a smaller exposure even though it passes through both threat envelopes consider Figure 4-11(a) which shows the LOS exposure of the



(a) Best Overall Score



(b) Population Average Score



(c) Trajectories

Figure 4-7: Scenario 5 Results

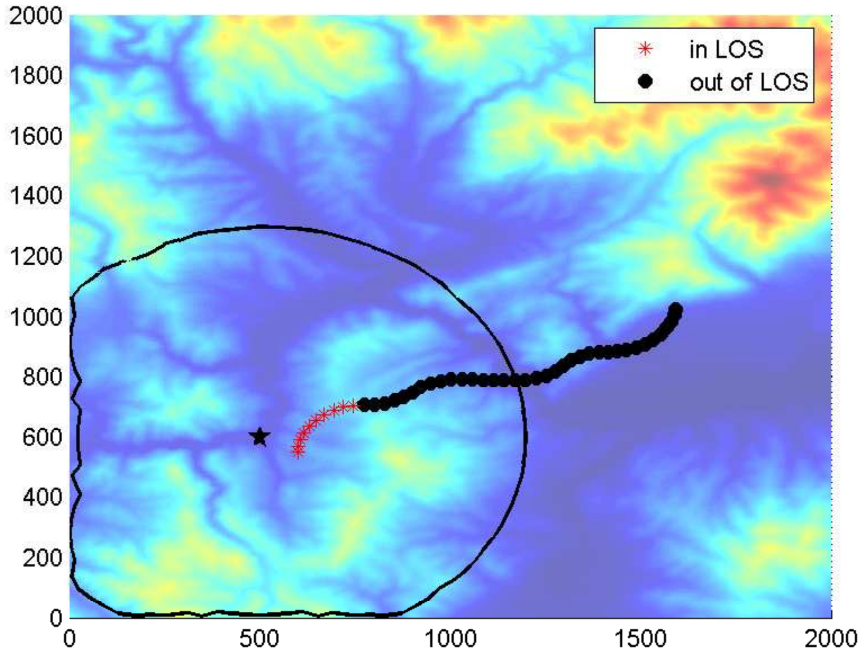


Figure 4-8: Scenario 5 Line-of-Sight Visualization

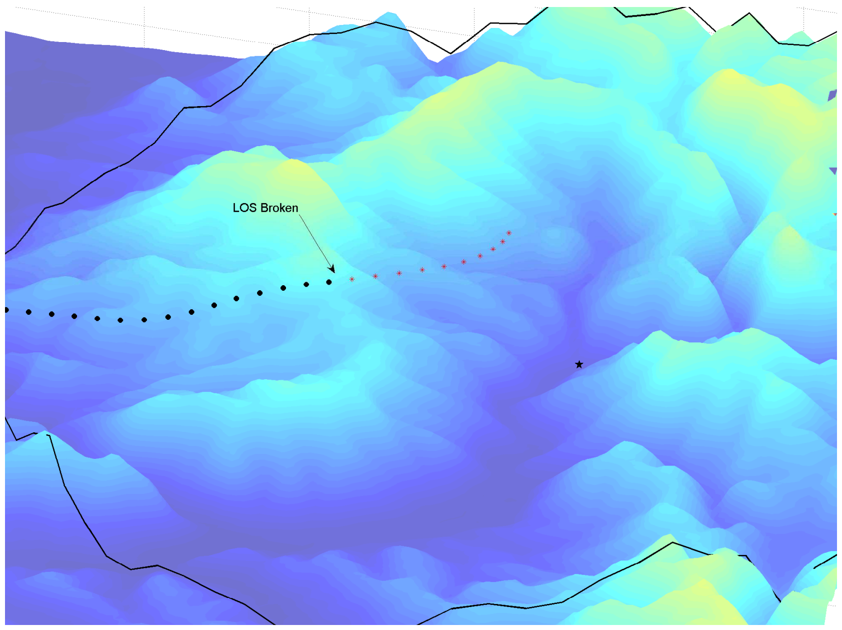
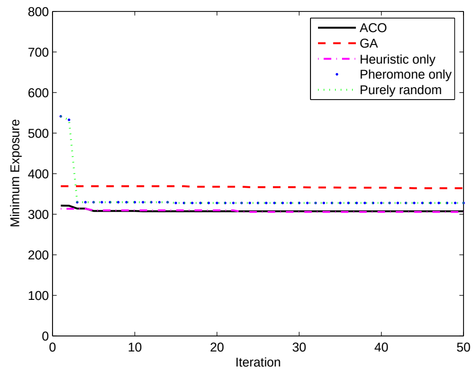
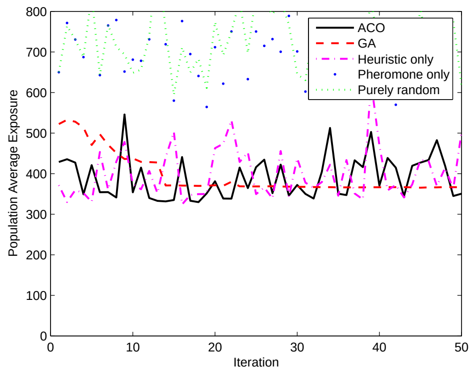


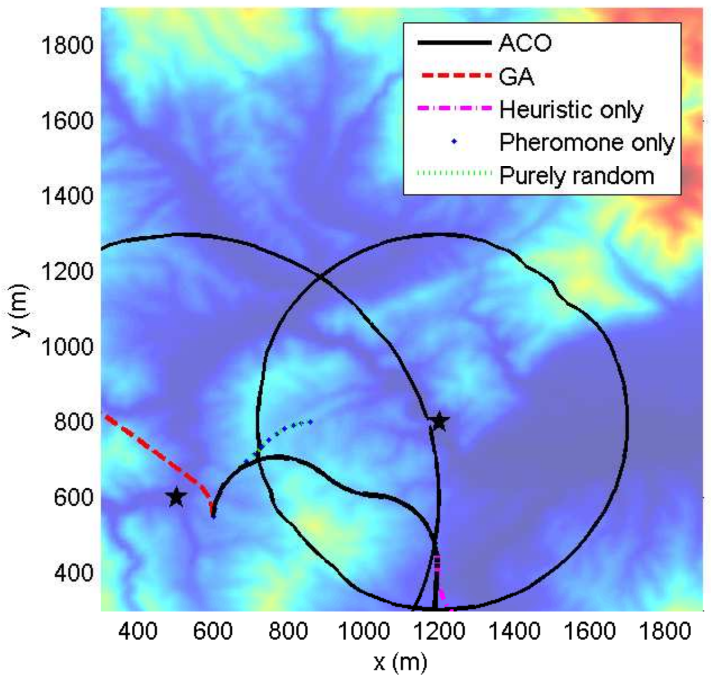
Figure 4-9: Scenario 5 Line-of-Sight Perspective



(a) Best Overall Score

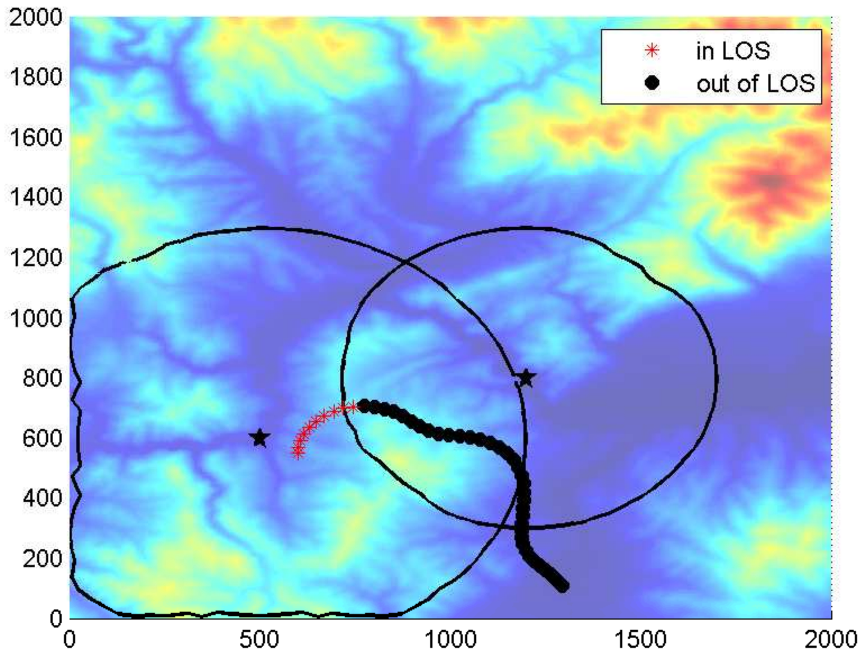


(b) Population Average Score

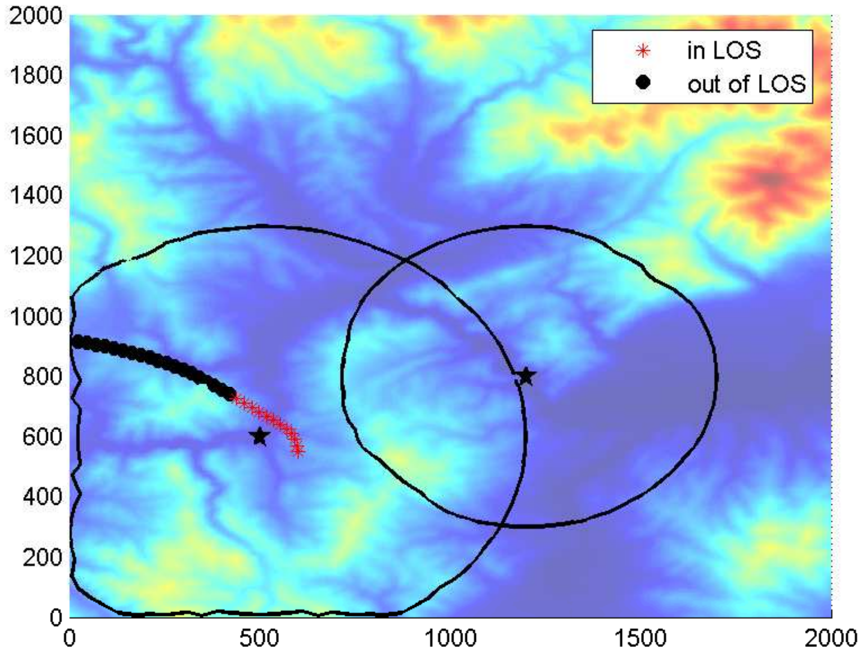


(c) Trajectories

Figure 4-10: Scenario 6 Results

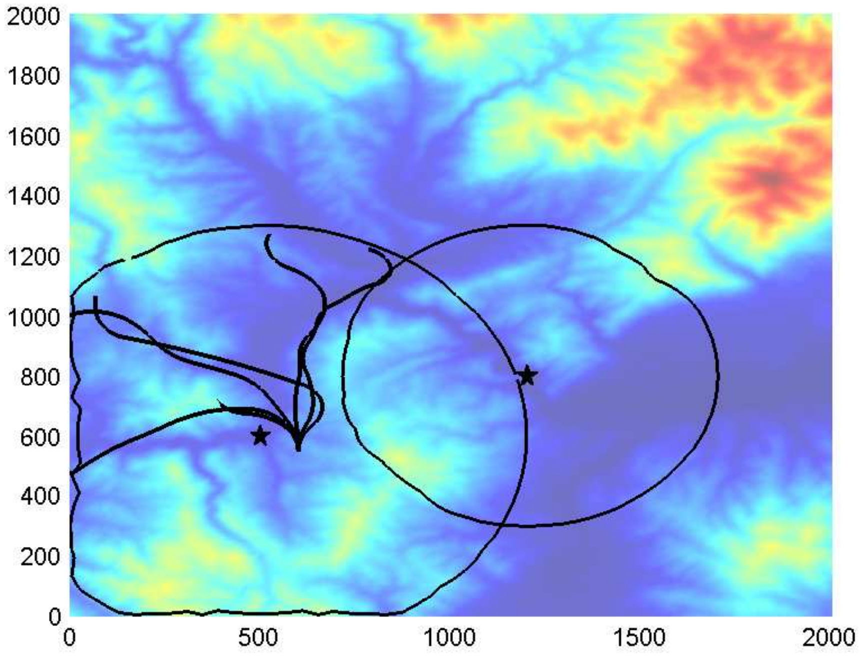


(a) ACO

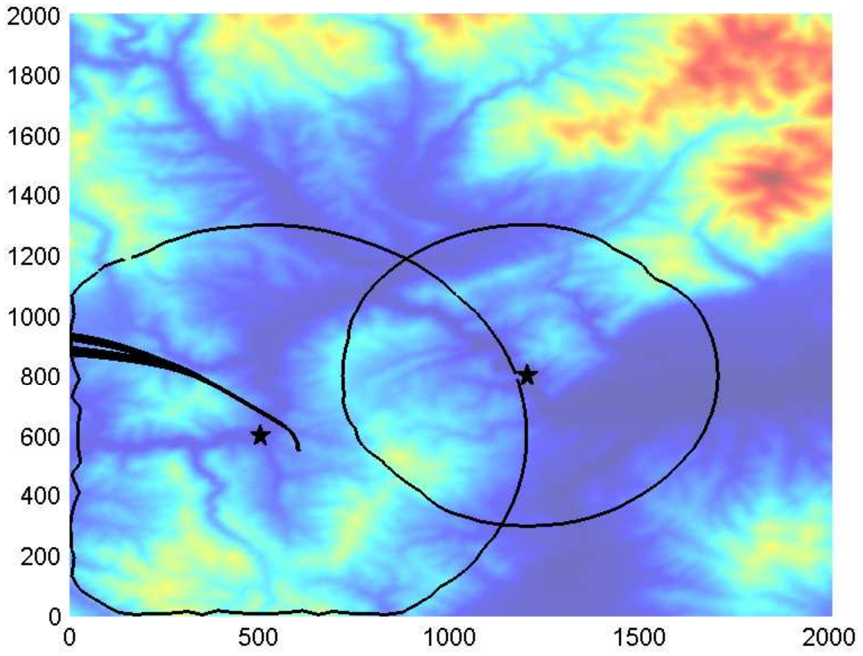


(b) GA

Figure 4-11: Scenario 6 Line-of-Sight Visualization



(a) Generation 5



(b) Generation 50

Figure 4-12: Genetic Algorithm Generation Comparison

ACO trajectory. Since the second threat is in a ravine, by flying close to the ridge the ACO trajectory is able to avoid LOS exposure with both threats. The GA is also able to break LOS (Figure 4-11(b)), but by flying closer to the first threat its exposure becomes much higher. One possible explanation for this behavior is a lack of exploration by the GA. Figures 4-12(a)–(b) show the population of solutions at generation 5 and 50 respectively. Notice that the GA has the tendency to stagnate toward later generations so little exploration takes place.

4.1.7 Scenario 7

Scenario 7 introduces the added difficulty of a multi-objective cost function. Here the UAV has the task of avoiding exposure to a single threat as much as possible, but it must still get near the goal point. This is meant to represent the case where a pop-up threat appears, but rather than abandoning whatever mission the vehicle was flying, it must reach the next waypoint despite the unexpected change in environment. The vehicle begins at $(x, y) = (600, 550)$ at 100 m AGL, heading north (positive y axis) with a velocity of 50 kts. The goal point is $(1000, 1600)$. A threat with a radius of 700 m is located at $(500, 600)$. The cost function used was $w_1 J_1 + w_2 J_2$ where J_1 is a minimum distance objective given by Equation 3.4 and J_2 is a minimum exposure objective given by Equation 3.6. The heuristic used is given by Equation 3.10.

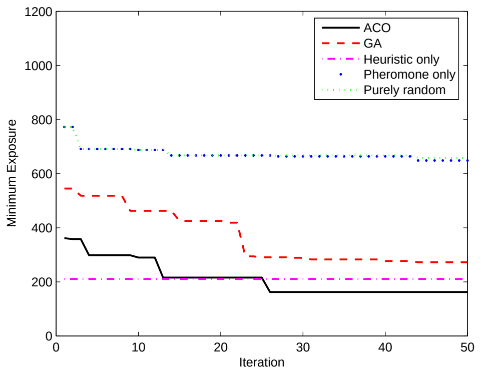
Again, both ACO and GA generate reasonable trajectories (Figure 4-13(c)). While the GA tries to circumvent the threat, it does not make it to the goal point. The ACO compromises by taking a straighter path through the threat envelope. This behavior can of course be controlled by changing the weighting on the two terms in the cost function. For this experiment, both weights were set to unity. From Figure 4-13(a) it appears that both the ACO and GA improve at roughly the same rate. The ACO starts with a better initial solution though, most likely due to the use of a heuristic. ACO also manages to use terrain masking to its advantage as seen in Figure 4-14(a).

4.1.8 Scenario 8

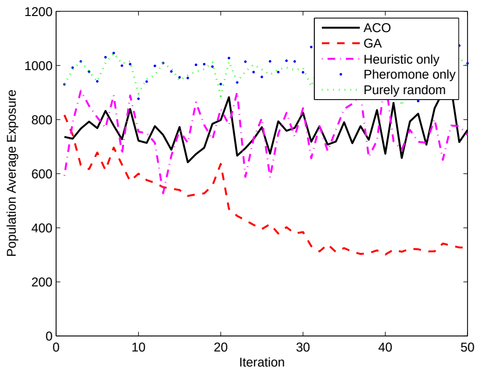
A slight extension to scenario 7, scenario 8 adds an extra threat to the environment. This is the full-blown multi-threat, multi-objective motion planning problem that is the basis of this thesis. The vehicle begins at $(x, y) = (600, 550)$ at 100 m AGL, heading north (positive y axis) with a velocity of 50 kts. The goal point is $(1000, 1600)$. A threat with a radius of 700 m is located at $(500, 600)$. An additional threat with a radius of 500 m is also present at $(1200, 800)$. Results are similar to those of scenario 7 (Figure 4-15).

4.2 Parameter Sensitivity

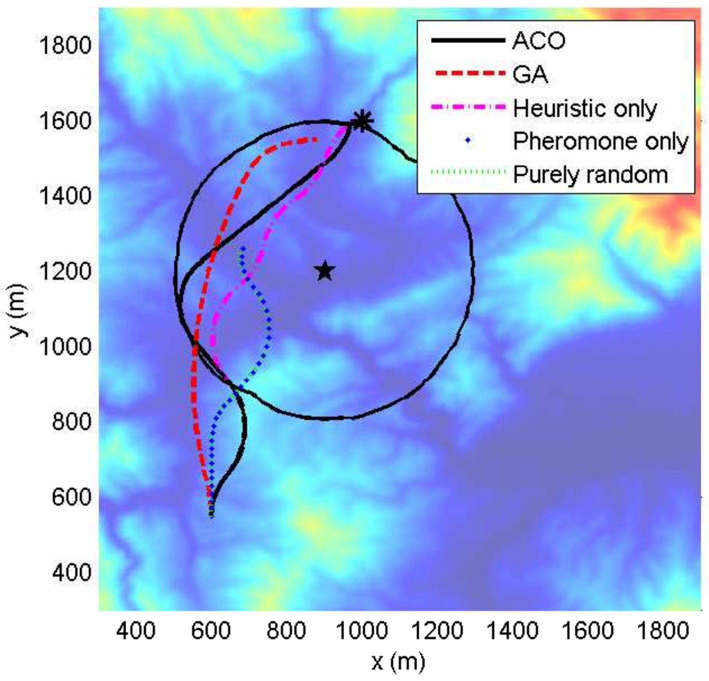
During implementation of the ACO algorithm, it is necessary to select several design parameters. The most notable among these are α, β, q_0, and τ_0. As mentioned



(a) Best Overall Score

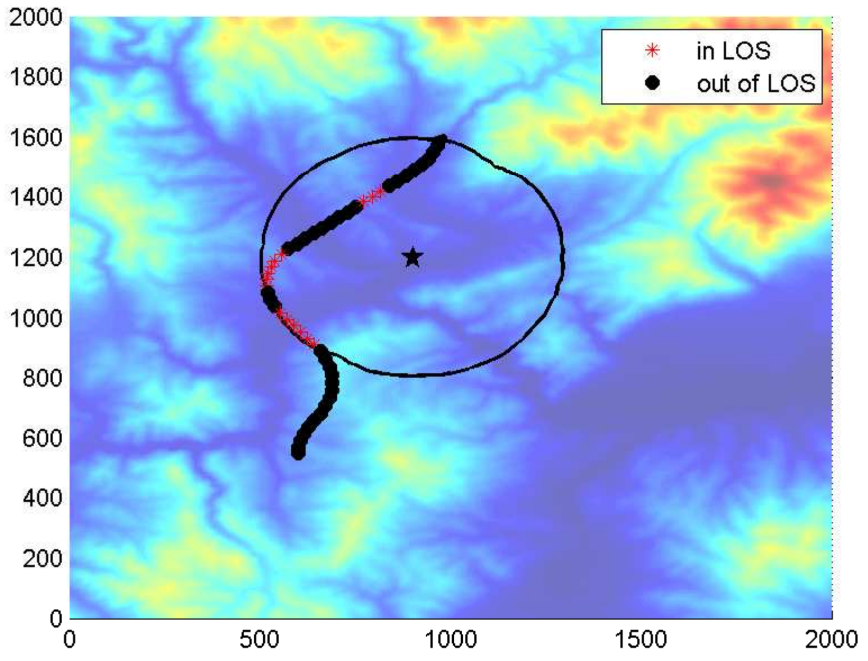


(b) Population Average Score

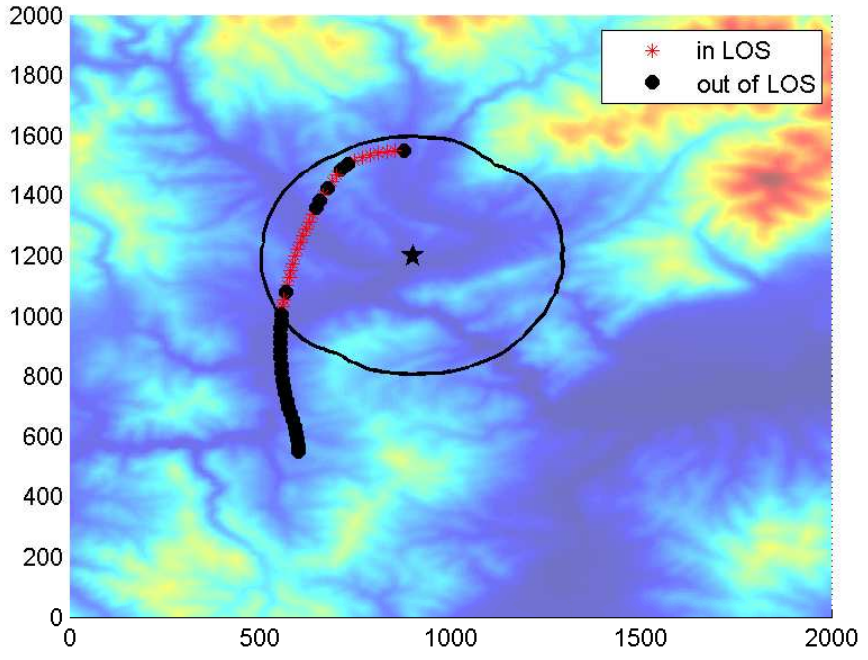


(c) Trajectories

Figure 4-13: Scenario 7 Results

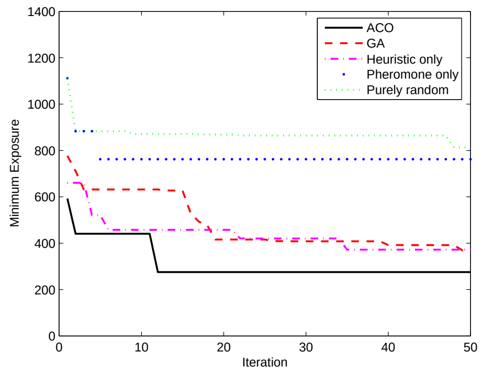


(a) ACO

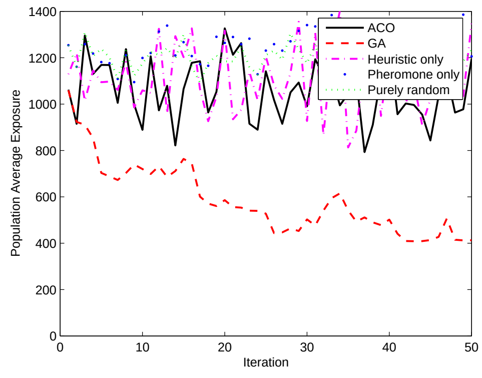


(b) GA

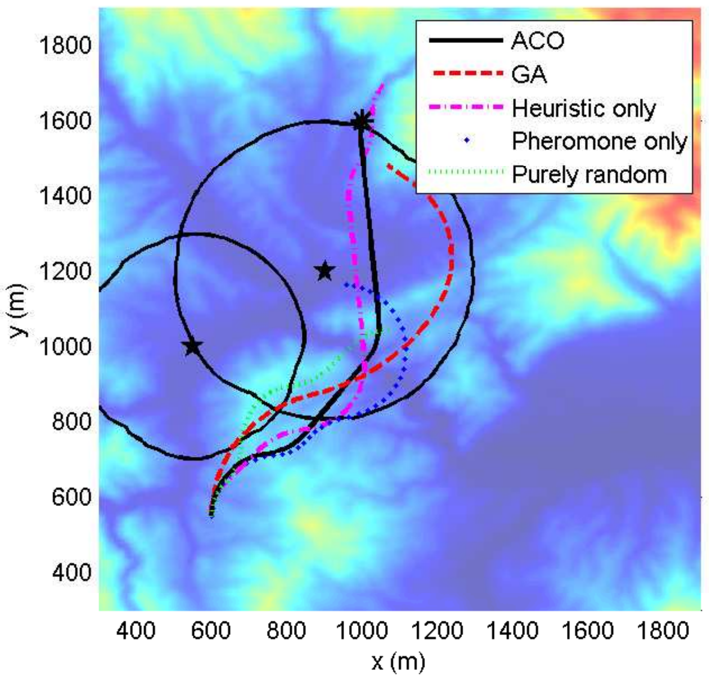
Figure 4-14: Scenario 7 Line-of-Sight Visualization



(a) Best Overall Score

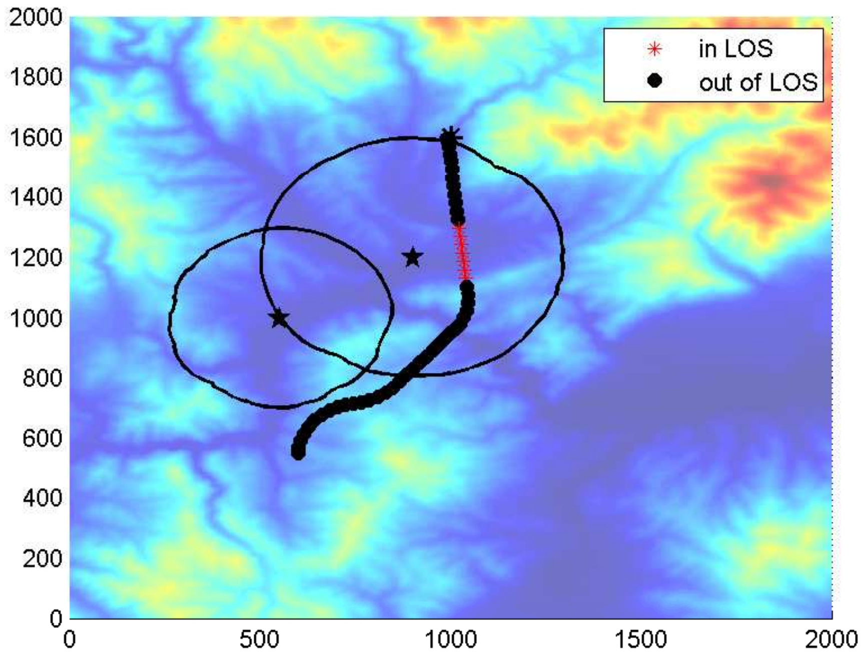


(b) Population Average Score

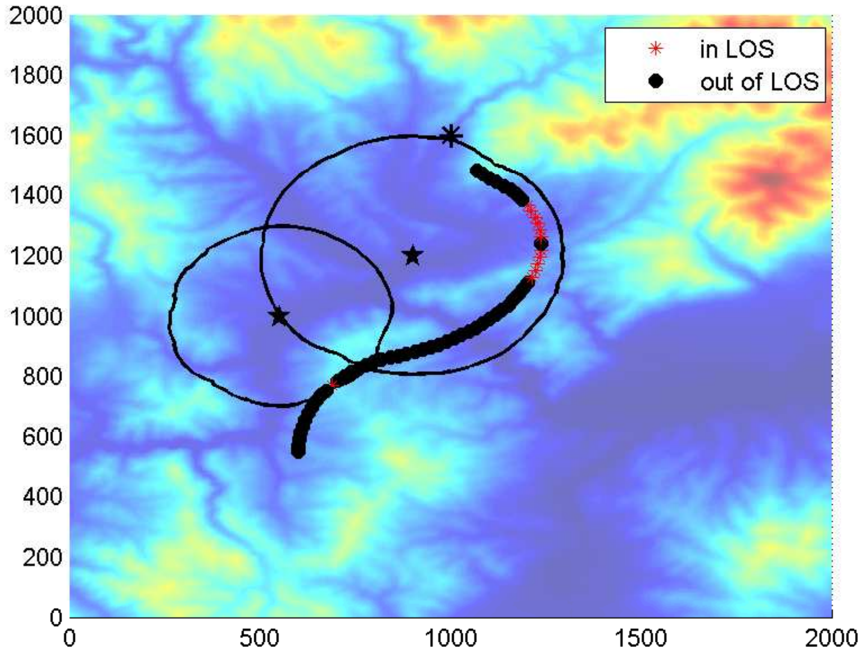


(c) Trajectories

Figure 4-15: Scenario 8 Results



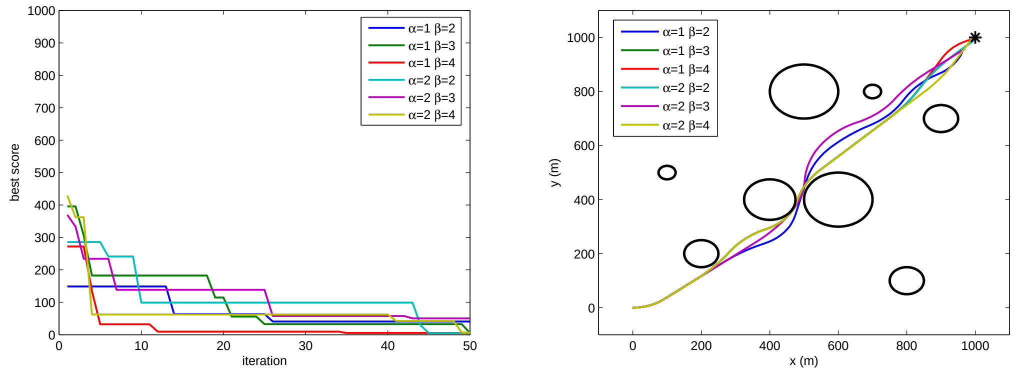
(a) ACO



(b) GA

Figure 4-16: Scenario 8 Line-of-Sight Visualization

in Section 3.4.4 q_0 and τ_0 were fixed at 0.5 and 10 respectively for each scenario. Reasonable values for each parameter were obtained from historical data [9]. In analyzing the properties of ACO as applied to motion planning problems, parameter sensitivity is a possible concern. As the specific motion planning that needs to be solved will, in general, not be known in advance the algorithm should be relatively insensitive to parameter selection. That is, it should be able to generate good solutions for a range of parameter values. Figure 4-17 shows the solutions produced by the



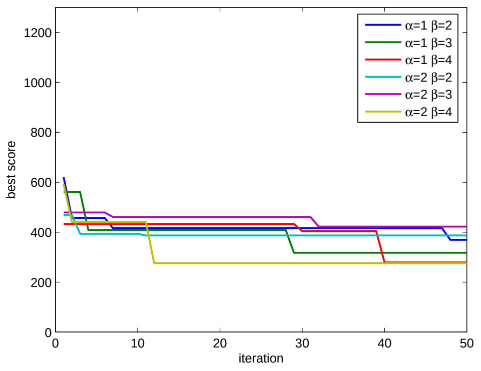
(a) Best Overall Score (b) Trajectories

Figure 4-17: Scenario 3 Parameter Comparison

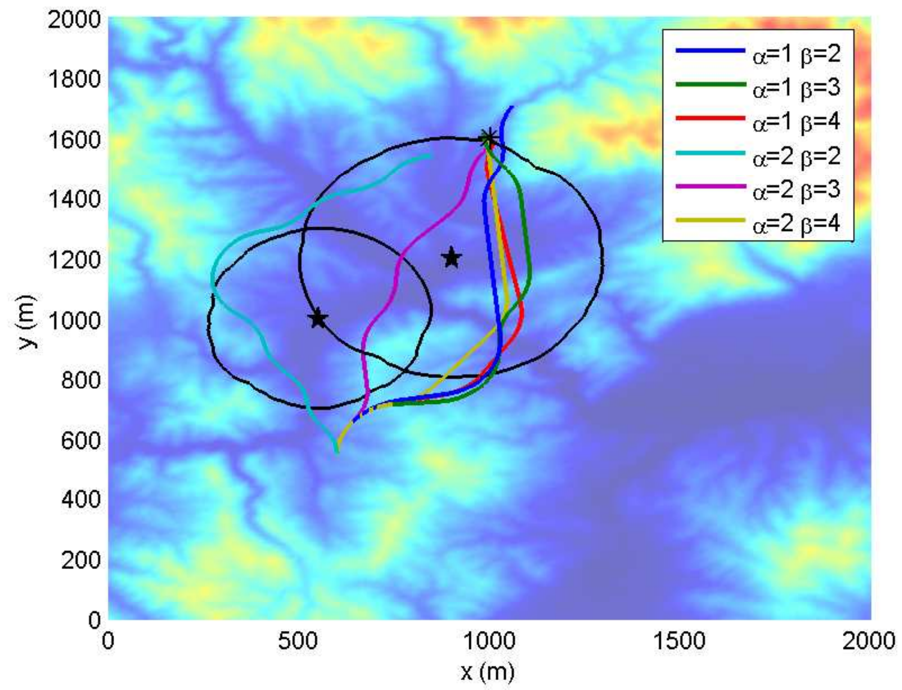
ACO algorithm for scenario 3 for various values of α and β. It would appear that in this case the algorithm is rather insensitive to parameter changes. Each selection produces nearly identical final trajectories (Figure 4-17(b)), and while there is a noticeable difference in convergence rate (Figure 4-17(a)), they all end up finding good solutions. Thus if the user needs a solution quickly, parameter selection becomes a more critical issue. The same parameter sweep was applied to scenario 8 (Figure 4-18) and the results are quite different. Not only does each selection produce a vastly different trajectory, the convergence rate and quality of the final trajectory are very different. It would appear that the complications added by the increased state-space size, and discontinuous nature of the objective function makes parameter selection a more critical issue. It is difficult to find a balance between α and β that perform well across all scenarios.

4.3 Runtime Performance

Since the eventual goal of this algorithm development is real-time performance some time must be spent analyzing the runtime characteristics of the ACO algorithm. Both the ACO and GA algorithms were run on a Pentium 4 3.2GHz with 1GB of RAM. In the 2D scenarios (1-3) the ACO had a maneuver library of 35 maneuvers. The average time for a single ACO generation was 33.4 ms, while the average time for a GA generation was 31.8 ms. In the 3D scenarios the ACO had a maneuver library

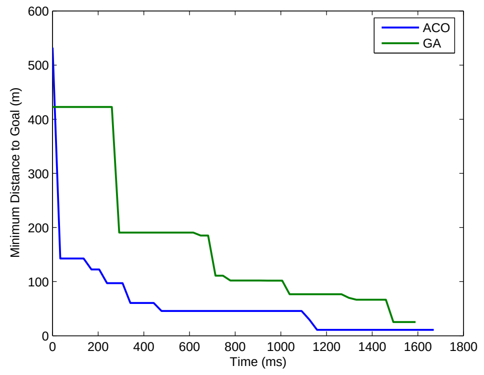


(a) Best Overall Score

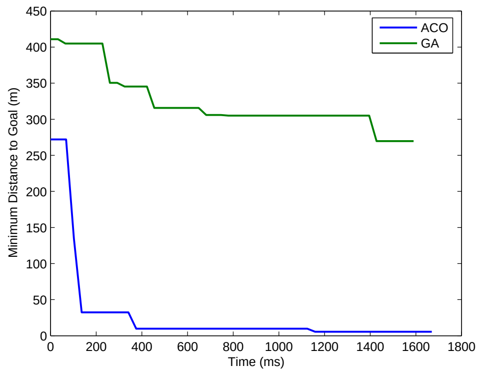


(b) Trajectories

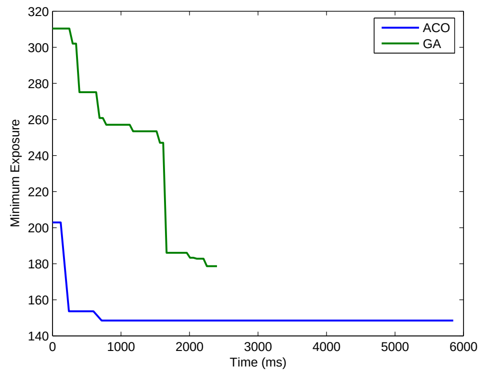
Figure 4-18: Scenario 8 Parameter Comparison



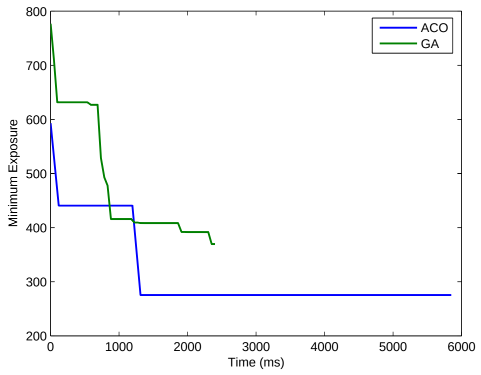
(a) Scenario 1



(b) Scenario 3



(c) Scenario 4



(d) Scenario 8

Figure 4-19: Runtime Performance of ACO and GA

of 144 maneuvers. In these cases the average time for a single ACO generation was 117 ms, while the average time for a GA generation was 48 ms. The reason for the discrepancy is most likely due to the pheromone maintenance of the ACO algorithm. A good deal of time must be spent updating and maintaining the pheromone values, as well as calculating the heuristic values for newly visited nodes. Clearly, as the number of maneuvers increases this maintenance becomes more costly. This is not the whole story though. Due to the faster convergence of ACO per generation and its better initial value the ACO algorithm is able to find comparable solutions in about the same amount of time as the GA. For example, in the same scenario it took ACO 1100 ms to find a solution with cost lower than 420, while the GA took 960 ms. Also, the ACO continued to show significant improvement after this point, while the gains from the GA were marginal. This can be seen in Figure 4-19 where best score has been plotted as a function of time. In most every case ACO is able to return a good solution faster than GA.

Chapter 5

Conclusions

5.1 Analysis

This section will present some more in-depth discussion and big picture ideas from the results. Recall that the objective of this thesis was to develop and validate a real-time motion planner for use in agile aerial vehicles. It would seem that this objective has been met. ACO was able to meet the requirements presented in Section 2.1. Good solutions were found for a wide range of motion planning problems. In each scenario presented, ACO outperformed GA and the other algorithms to some degree. Some time should be spent analyzing why the ACO performed like it did, and any benefits or detriments of the algorithm that were not known before. As discussed earlier, the main difference between ACO and GA is the way in which solutions are constructed and information passed from generation to generation. The hypothesis was that since ACO builds solutions iteratively rather than using a crossover style method, it is better suited to trajectory generation problems. This seems to be confirmed by the results, but it is not the whole story. Several key issues that warrant further discussion are heuristic influence, parameter selection, effect of state-space size, and stagnation.

It would seem that the heuristic plays a large role in ACO solution quality. Indeed, even when the algorithm was originally developed and applied to the TSP a good heuristic was needed to be competitive with existing methods [9]. This trend is evident in the results where the ACO solutions very closely mimic that of the heuristic. This strong heuristic contribution is something of a double-edged sword. If a good heuristic can be developed from problem-specific information then ACO can perform very well. Computing the heuristic values though may result in an undesirable increase in runtime. In the absence of a good heuristic ACO has little hope of performing well.

A similar conclusion can be drawn for other randomized search methods. Indeed, with any of these general heuristic methods starting solution generation with problem-specific knowledge will likely result in better performance than if no prior information is given. If heuristic information were incorporated into GA it would likely perform better. Using heuristic information in a GA could take several forms. The population could be initialized by selecting maneuvers weighted by heuristic values, rather than purely random. Also, the mutation operator could be modified so that mutations toward maneuvers with higher heuristic values are more likely.

The parameter selection aspect of ACO is perhaps the most troublesome. There appears to be no good way to select parameters such as α, β, τ_0, etc. other than trial-and-error. It was seen that for simple problem instances these parameters had little effect on the outcome. For large-scale problems though there was a staggering difference in solutions obtained. When a 'good' set of parameters is selected ACO can outperform existing methods, but in the absence of reasonable parameter values algorithm performance can be poor.

It would seem that ACO was indeed able to handle the large state-space presented by the motion planning problem. It should be noted though that this ability is most likely due to the use of a heuristic. The effect of pheromone deposit by a single agent is reduced as state space size increases. For example, consider the case where m maneuvers are available in the library, and a trajectory is n maneuvers long. This leads to potentially m^n states, and as many branches to deposit pheromone on. An agent will only affect the pheromone on n of these branches. Compare this to the TSP case where a fully connected graph with n nodes has $O(n^2)$ edges to deposit pheromone on. The two most obvious solutions to this problem are using more agents and running the algorithm for more generations. Each of these will result in an increase in runtime. A more sophisticated solution could be to use 'pheromone smearing' (Section 5.2.1) so that an agent affects more branches than just those traversed. It is essentially a form of state aggregation in which similar states and maneuvers are grouped so that pheromone update on one member of the group would affect the others.

Stagnation is also a possible concern. The plots of best overall solution seem to indicate that ACO quickly levels off and shows little improvement during later generations. But Figures 4-2(a)–(b) show that the population is not converging toward a single region. The GA shows an opposite trend. There is steady improvement in the best solution found but the population converges on a very narrow region during later generations. One possible explanation is that ACO quickly finds a good solution, but fails to exploit this knowledge in later generations. That is, the ACO algorithm is exploring too much and not making use of information of good solution regions. If this is the case, one possible solution is adjusting the parameters so that exploitation of known solutions is heavily favored. This can be accomplished by selecting larger values for q_0, α, and the amount of pheromone deposited by an agent.

In general, ACO is well suited to agile motion planning problems, and is capable of outperforming existing methods. Particularly in the case where the environment is cluttered and obstacle avoidance becomes a primary concern, ACO can perform significantly better than GA. This is due to the iterative manner in which ACO constructs trajectories. While GA has to discard many possible solutions due to the tendency of the crossover operator to produce infeasible solutions, ACO does not need to discard any solutions and is able to avoid infeasible trajectories during construction. This performance is dependent on existence of a good heuristic. If no such heuristic is available, a GA is likely the more appropriate choice.

5.2 Future Work

While the ACO algorithm performed well, there are still possibilities for improvement. This section presents some ideas for future areas of research.

5.2.1 Pheromone Maintenance

Pheromone maintenance is the most critical aspect of any ACO implementation. The scheme used in this thesis is described in Section 3.4.2. Many other options are available though. Currently, when an agent deposits pheromone, and equal amount is deposited along each branch of the decision tree. One possible modification is to weight pheromone distribution so that larger amounts get deposited at the earlier branches. This has the potential effect of more strongly affecting the initial maneuver selections. This could be useful in the case where the first steps made are also the most critical, such as the extreme case of choosing to go left or right around a threat. Another possible modification is that of 'pheromone smearing'. As discussed in Section 5.1 the large state space of the motion planning problem can result in each agent having a reduced effect on subsequent generations. With 'pheromone smearing' rather than depositing pheromone only on the maneuver branch selected, each agent would deposit pheromone on several maneuvers. For example, the benefit from making a left turn of 20 degrees is probably similar to that from making a 15 degree turn. As such, when an agent selects the 20 degree turn as a maneuver and later updates the pheromone at that branch, it can also update the pheromone along the 15 degree turn branch. By defining a metric between maneuvers it may be possible to have each agent adjust the pheromone levels on the maneuver it selected as well as on 'similar' maneuvers.

5.2.2 ACO Extensions

Several extensions to the ACO formulation are also possible. The fact that ACO is a discrete optimization scheme is potentially a limiting factor. One possible way of extending it to continuous optimization is to use a continuous parameterization of maneuvers and Gaussian kernels as the pheromone deposits [30]. This would require the development of a pheromone look-up function $F : \mathcal{X} \rightarrow \mathcal{P}$ which maps states, $\mathbf{x} \in \mathcal{X}$, to probability distributions over the set of available maneuvers, $\mathcal{P}$. Another possible way of extending ACO to continuous spaces is to integrate a local search algorithm. That is, after an agent has constructed a solution, use a local search scheme such as a gradient descent method to make minor parameter modifications to each maneuver.

Another idea to increase algorithm efficiency is to implement a receding horizon version of ACO. That is, rather than planning for a large time horizon, plan only for a small one with the intention of replanning when the short-term horizon is reached. ACO seems to lend itself well to receding horizon methods since the pheromone information gained during one planning period can be used for subsequent planning periods. In terms of implementation this can be accomplished by starting the search from a different node in the decision tree.

5.2.3 Parameter Optimization

The results in Section 4.2 are disconcerting. More work needs to be done to determine if it is possible to optimize parameter selection, or how to decrease the sensitivity of ACO to parameter changes. It is possible that this parameter sensitivity is due in part to large state spaces. This seems to be supported by Figures 4-17 and 4-18 where a simple 2D search problem was unaffected by parameter changes. As such, modifications such as 'pheromone smearing' or state aggregation might reduce sensitivity.

Another 'parameter' that should be further investigated is the maneuver library used. For this thesis a roughly uniform distribution of maneuvers was used within the flight envelope. It is possible that there are better sets of maneuvers to use. In the case of parameterized maneuvers one possible measure of quality could be the size of the reachable space of the maneuver primitive in a given time horizon. Motion capture from human pilots is also a possible source of high-quality maneuver primitives.

Bibliography

[1] J.M. Ahuactzin, El-Ghazali Talbi, Pierre Bessiere, and Emmanuel Mazer. Using Genetic Algorithms for Robot Motion Planning. In *10th European Conference on Artificial Intelligence*, pages 671–675, 1992.

[2] Sabi Asseo. In-flight Replanning of Penetration Routes to Avoid Threat Zones of Circular Shapes. In *Proceedings of IEEE 1998 National Aerospace and Electronics Conference*, Dayton, OH, July 1998.

[3] Randal Beard, Timothy McLain, Michael Goodrich, and Erik Anderson. Coordinated Target Assignment and Intercept for Unmanned Air Vehicles. *IEEE Transactions on Robotics and Automation*, 18(6):911–922, 2002.

[4] George Bilchev and I.C. Parmee. The Ant Colony Metaphor for Searching Continuous Design Spaces. In *Proceeding of the AISB Workshop on Evolutionary Computation*, University of Sheffield, UK, April 1995.

[5] Scott Bortoff. Path Planning for UAVs. In *Proceedings of the 2000 American Control Conference*, Chicago, IL, June 2000.

[6] Bernd Bullnheimer, Richard Hartl, and Christine Strauss. A New Rank Based Version of the Ant System — A Computational Study. *Central European Journal of Operations Research and Economonics*, 7(1):25–38, 1999.

[7] Chris Dever. *Parametrized Maneuvers for Autonomous Vehicles.* PhD thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, September 2004.

[8] Atilla Dogan. Probabilistic Path Planning for UAVs. In *2nd AIAA Unmanned Unlimited Systems, Technologies, and Operations*, San Diego, CA, September 2003.

[9] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization.* The MIT Press, Cambridge, Massachusetts, 2004.

[10] Emilio Frazzoli. *Robust Hybrid Control for Autonomous Vehicle Motion Planning.* PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2001.

[11] Luca Gambardella and Marco Dorigo. An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem. *INFORMS Journal on Computing*, 12(3):237–255, 2000.

[12] Luca Gambardella, Éric Taillard, and Giovanni Agazzi. MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. Technical Report 06-99, Dalle Molle Institute for Artificial Intelligence, Lugano, Switzerland, 1999.

[13] Datta Godbole, Tariq Samad, and Vipin Gopal. Active Multi-Model Control for Dynamic Maneuver Optimization of Unmanned Air Vehicles. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 2000.

[14] David Goldberg. *Genetic Algorithms: Search, Optimization and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989.

[15] John Hansman and Roland Weibel. Human and Automation Integration Considerations for UAV Systems. http://tinyurl.com/f5lr8, 2004.

[16] John Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, Michigan, 1975.

[17] Mansoor Karimifar and Jeffrey Chien. Ant Conversation - An Enhancement of ACO. In *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, New York, August 2004.

[18] Jongwoo Kim and James Ostrowski. Motion Planning of Aerial Robot Using Rapidly-exploring Random Trees with Dynamic Constraints. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, September 2003.

[19] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, Massachusetts, 1991.

[20] Steven LaValle and James Kuffner Jr. Randomized Kinodynamic Planning. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, Detriot, MI, May 1999.

[21] Peter Marbach and John Tsitsiklis. Simulation-Based Optimization of Markov Reward Processes. *IEEE Transactions on Automatic Control*, 46(2):191–209, February 2001.

[22] Daniel Merkle, Martin Middendorf, and Hartmut Schmeck. Ant Colony Optimization for Resourse-Constrained Project Scheduling. *IEEE Transactions on Evolutionary Computation*, 6(4):333–346, aug 2002.

[23] Office of the Secretary of Defense Unmanned Aircraft Systems. Unmanned Aircraft Systems Roadmap. http://www.acq.osd.mil/uas/, 2005.

[24] Ryan L. Pettit. Low Altitude Threat Evasive Trajectory Generation for Autonomous Vehicles. Master's thesis, Massachusetts Institute of Technology, Aerospace Engineering, July 2004.

[25] Boeing News Release. Boeing X-45 Unmanned Aircraft Demonstrates Autonomous Capability. http://tinyurl.com/epeof, 2005.

[26] Michael Rice. Computational Intelligence Algorithms for Optimized Vehicle Routing Applications in Geographic Information Systems. Thesis Proposal, 2004.

[27] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Upper Saddle River, New Jersey, 2003.

[28] John Sauter, Robert Matthews, H. Van Dyke Parunak, and Sven Brueckner. Evolving Adaptive Pheromone Path Planning Mechanisms. In *Proceedings of the First International Conference on Autonomous Agents and Multi-Agent Systems*, pages 434–440, Bologna, Italy, 2002. ACM.

[29] Tom Schouwenaars, Bernard Mettler, Eric Feron, and Jonathan How. Hybrid Model for Trajectory Planning of Agile Autonomous Vehicles. *Journal of Aerospace Computing, Information, and Communications*, 1(12):629–651, 2004.

[30] Krzysztof Socha. ACO for Continuous and Mixed-Variable Optimization. In *ANTS Workshop*, 2004.

[31] Thomas Stützle and Holger Hoos. $\mathcal{MAX}-\mathcal{MIN}$ Ant System. *Journal of Future Generation Computer Systems*, 16:889–914, 2000.

[32] Gregory Toussaint, Tamer Başar, and Francesco Bullo. Motion Planning for Nonlinear Underactuated Vehicles Using H^∞ Techniques. In *Proceedings of the American Control Conference*, Arlington, VA, June 2001.