

# CONSTRAINED DYNAMIC ROUTE PLANNING FOR UNMANNED GROUND VEHICLES

Anthony Stentz  
Robotics Institute  
Carnegie Mellon University  
Pittsburgh, PA 15213  
tony@cmu.edu

## ABSTRACT

Unmanned Ground Vehicles are tasked with negotiating off-road terrain while satisfying certain objectives, such as maintaining cover and concealment and arriving at a phase line by a designated time. The problem is complicated by the fact that the terrain is typically not known completely in advance, rendering pre-planned routes useless if selected passageways are obstructed. In this situation, the UGV must re-plan the route based on the new sensor information acquired, in such a way that it still satisfies the constraints and objectives. This paper presents a dynamic route planner that rapidly re-plans in response to new information. The planner represents some objectives as soft (e.g., maximizing concealment) and others as hard (e.g., arriving at a phase line on time) and optimizes the soft objectives as much as possible while still meeting the hard. The planner was tested on a simulated scout mission requiring a stealthy traverse with a phase line constraint, and the results are presented.

## 1. INTRODUCTION

Unmanned Ground Vehicles (UGV's) have the potential for performing a large number of tactical behaviors, including road, area, and zone reconnaissance. Fundamental to these behaviors is basic navigation, that is, moving from one point to another on the terrain. To move from A to B, the UGV selects a route that minimizes some objective function, such as distance travelled, energy consumed, risk incurred, and intelligence gained. Often, the set of feasible solutions is limited by one or more constraints, which can be local or global. A local constraint is evaluated at a single step in the solution, such as the constraint that a UGV must avoid all obstacles in the route. A global constraint applies to the entire solution, or at least a large portion of it, such as the constraint that a UGV must reach its goal within a fixed amount of time.

For example, consider the following problem. The UGV is tasked with finding the stealthiest collision-free path to the goal. A stealthy path is one that moves along the perimeter of terrain features to minimize visibility. Additionally, the UGV must reach the goal within a fixed amount of time. To optimize its objective function, the UGV favors paths that hop from one area of concealment to another and avoid cutting across open areas. But these longer paths are precisely the ones that are likely to violate the global time constraint.

Assuming a feasible and optimal solution can be produced, it may need to be modified as the "plan" is exe-

cuted. In the example given, the UGV may detect unknown obstacles enroute, requiring it to re-plan part or all of the remaining path to the goal. Re-planning from scratch for each such occurrence can be impractical if the planning time is long.

Exhaustive search can be used to find an optimal solution by enumerating and evaluating the possibilities. Local constraints are trivial to handle, but global constraints are more problematic. Another variable or dimension can be added to the search space that tracks each global constraint and prunes the search when one is violated, but this approach can dramatically increase the complexity of the search. While exhaustive search may be acceptable for the initial, off-line plan, it is generally unacceptable for re-planning during execution, since ideally the UGV waits for the re-planner to complete before proceeding.

The most comprehensive and formal treatment of the problem in the AI literature is Logan's ABC algorithm (Logan 1998). ABC is a generalized A\* search capable of handling multiple global constraints of a variety of forms. For a limited class of these constraints, the algorithm is both optimal and complete. Given that constraints are tracked in each state and non-dominated paths are retained, we expect ABC to exhibit the same complexity problems as the exhaustive search described above, since the problem is NP-complete (Garey and Johnson 1979). The results presented are for a small planning space with two constraints and no run times are provided.

A similar problem, called Quality of Service (QoS), is addressed in the communications and network literature. QoS addresses the problem of shortest-path routing through a data network subject to constraints like bandwidth, delay, and jitter. In most cases, the costs are additive and the constraints are inequalities. The A\*Prune algorithm (Liu and Ramakrishnan 2001) solves the QoS problem and is optimal and complete, but it also exhibits exponential growth. To circumvent the complexity of exact solutions, the QoS literature is replete with polynomial algorithms for producing approximate solutions (Jaffe 1984) (Hassin 1992) (Korkmaz et al. 2000) (Korkmaz and Krunz 2001).

Noticeably absent from the literature is a computationally efficient *re-planner* for globally constrained problems with unknown, uncertain, or changing cost data. The problem of real-time, optimal re-planning was first addressed by Stentz (Stentz 1994) (Stentz 1995) with the D\* (Dynamic A\*) algorithm. D\* produces an initial plan using known, assumed, and/or estimated cost data, commences execution of the plan, and then rapidly re-plans enroute each time new cost information arrives. By itself,

D\* optimizes a single cost metric and does not offer an efficient way to optimize globally constrained problems.

This paper describes the Constrained D\* (CD\*) algorithm (Stentz 2002), which produces a resolution-optimal solution for problems with a global constraint. We begin by developing CA\* (Constrained A\*), an efficient, resolution optimal planner similar to Korkmaz's work in QoS (Korkmaz et al. 2000). We then transform CA\* into CD\* to make an efficient re-planner, measure its speed through experiments in simulation, and conclude with a comprehensive example that illustrates its use.

## 2. ALGORITHM DESCRIPTIONS

The Constrained D\* (CD\*) algorithm is a computationally efficient planning and re-planning algorithm. CD\* realizes this efficiency by avoiding the addition of dimensions to the search space to handle a global constraint; instead, the algorithm incorporates the constraint in the objective function itself. The constraint is multiplied by a weight, which the algorithm adjusts using a binary search algorithm. The search problem is solved end to end multiple times, with CD\* adjusting the weight from one iteration to the next to converge to an optimal solution that satisfies the global constraint.

To plan in real time, CD\* saves each stage of the binary search process. When new information arrives during execution, CD\* repairs the first stage. If the weight selection for the second stage is unchanged, CD\* repairs the stage and continues. CD\* re-plans from scratch at stage  $i$  only when stage  $i-1$  selects a different weight. In practice, this occurs infrequently and is the basis for CD\*'s speed. We start by introducing CA\*, an efficient algorithm capable of finding a resolution optimal solution that satisfies the global constraint, then we show how this algorithm can be modified to efficiently re-plan (CD\*). The reader is referred to earlier work (Stentz 2002) for proofs of algorithm correctness.

### 2.1 The Constrained A\* Algorithm

Let  $f_0(\circ)$  be the objective function to optimize, and let  $f_1(\circ)$  be the global constraint to satisfy. Without a loss of generality, assume that  $f_0(\circ)$  is optimized when it is minimized. Assume that  $f_1(\circ)$  is satisfied when its value is less than or equal to some constant  $K$ . Both functions are defined across candidate solutions to the problem,  $X$ . For the example given in the introduction,  $X$  is a path (sequence of grid cells) leading from the goal state to the start state in a two-dimensional grid,  $f_0(\circ)$  is the sum of the visibility costs along  $X$ , and  $f_1(\circ)$  is the required time to reach the goal along  $X$ .

The objective for CA\* is to find the  $X$  that minimizes  $f_0(\circ)$  such that  $f_1(X) \leq K$ . To do this, CA\* minimizes a composite function  $f(\circ)$  with the following form:

$$\text{Equation 1: } f(X, w) = f_0(X) + wf_1(X)$$

where  $w$  is a non-negative weight that sets the balance between minimizing the objective function and satisfying the global constraint. CA\* picks an initial value for  $w$  and minimizes  $f(\circ)$ . If the global constraint is violated (i.e.,  $f_1(X) > K$ ) then CA\* increases  $w$  to steer the solution away from those states that violate the constraint, and re-starts the search to minimize  $f(\circ)$  again. If the global constraint is satisfied (i.e.,  $f_1(X) \leq K$ ), then CA\* reduces  $w$  to find a lower value for  $f_0(\circ)$  that still satisfies the constraint. The algorithm repeats until it finds a value for  $w$  that is just large enough to avoid violating the global constraint. The corresponding  $X$  and  $f_0(\circ)$  are the optimal solution and optimal cost, respectively, that satisfy the constraint.

To minimize the number of iterations required to find the appropriate  $w$ , CA\* uses binary search. If  $f(\circ)$  is a dynamic programming function, then A\* can be used to find the optimal  $X$ . The notation  $X = A^*(f(\circ), w, G)$  means that A\* is called on a graph  $G$  and returns the optimal  $X$  for a given objective function  $f(\circ)$  and weight value  $w$  (NOPATH is returned if no solution exists).  $N$  is the number of stages in the binary search, and  $(w_{min}, w_{max})$  is a pair of weights that bracket the optimal, constrained solution. The complete algorithm is given below:

```

L1   $X \leftarrow CA^*(f(\circ), K, w_{min}, w_{max}, N, G)$  :
L2   $X_{min} \leftarrow \emptyset$ ;  $X_{max} \leftarrow \emptyset$ 
L3  for  $i \leftarrow 1$  to  $N-1$ 
L4     $w \leftarrow (w_{max} + w_{min})/2$ 
L5     $X \leftarrow A^*(f(\circ), w, G)$ 
L6    if  $X = NOPATH$  then return NOPATH
L7    if  $f_1(X) \leq K$  then
L8       $w_{max} \leftarrow w$ ;  $X_{max} \leftarrow X$ 
L9    else
L10      $w_{min} \leftarrow w$ ;  $X_{min} \leftarrow X$ 
L11  if  $X_{min} = \emptyset$  then
L12     $X_{min} \leftarrow A^*(f(\circ), w_{min}, G)$ 
L13  if  $X_{min} = NOPATH$  then return NOPATH
L14  if  $f_1(X_{min}) \leq K$  then return HIGHRANGE
L15  if  $X_{max} = \emptyset$  then
L16     $X_{max} \leftarrow A^*(f(\circ), w_{max}, G)$ 
L17  if  $f_1(X_{max}) > K$  then return LOWRANGE
L18  return  $X_{max}$ 
L19 end

```

The CA\* algorithm finds the optimal solution for objective function that satisfies the global constraint to within a weight error of  $(w_{max} - w_{min})/2^{N-1}$ . If  $w_{max}$  is too small, CA\* returns *LOWRANGE*. If  $w_{min}$  is too large, CA\* returns *HIGHRANGE*. For such cases, the algorithm can be run again with an adjusted weight range.

### 2.2 The Constrained D\* Algorithm

The problem with CA\* is that it must re-plan from scratch whenever a new piece of cost information arrives (i.e.,  $G$  changes). For example, a UGV may discover that a passageway is unexpectedly blocked, thereby requiring the UGV to re-plan the remaining path to the goal. The algo-

rithm  $D^*$  is very efficient at re-planning whenever new information arrives (Stentz 1995). The notation  $(X, G) = D^*(f^\circ, w, G, \Delta)$  means that  $D^*$  is called on a graph  $G$  and returns the optimal  $X$  for a given function  $f^\circ$  and weight value  $w$  ( $NOPATH$  is returned if no solution exists). At first invocation,  $D^*$  runs in time comparable to  $A^*$ , as it produces the initial, optimal solution. The parameter  $\Delta$  represents changes to  $G$  (e.g., costs or connectivity). Initially,  $G$  is set to  $\emptyset$  and  $\Delta$  is set to the entire graph itself. Subsequent calls to  $D^*$  on  $G$  with changes  $\Delta$  produce a new optimal solution  $X$  and an updated graph  $G$  which incorporates the changes. This updated solution can be calculated very quickly—it is much faster than calling  $A^*$  on the modified graph. The difference, however, is in run time only. Both approaches produce the same solution, barring ties. If  $D^*$  is called with a new weight  $w$ , it is equivalent to changing every edge cost in  $G$ . Thus, the algorithm must re-plan from scratch, and the run time for that stage is comparable to  $A^*$ .

The complete algorithm for  $CD^*$  is given below.  $CD^*$  is similar to  $CA^*$ , except that it uses  $D^*$  instead of  $A^*$ , and it maintains an array of  $N$  graphs  $G[1..N]$ , one for each weight value  $w$  explored by the algorithm. Only the weights differ; the cost and connectivity information for each graph are identical.

```

L1   $(X, G[1..N]) \leftarrow CD^*(f^\circ, K, w_{min}, w_{max}, N, G[1..N], \Delta)$  :
L2   $X_{min} \leftarrow \emptyset$ ;  $X_{max} \leftarrow \emptyset$ 
L3  for  $i \leftarrow 1$  to  $N - 1$ 
L4     $w \leftarrow (w_{max} + w_{min})/2$ 
L5     $(X, G[i]) \leftarrow D^*(f^\circ, w, G[i], \Delta)$ 
L6    if  $X = NOPATH$  then return  $NOPATH$ 
L7    if  $f_1(X) \leq K$  then
L8       $w_{max} \leftarrow w$ ;  $X_{max} \leftarrow X$ 
L9    else
L10      $w_{min} \leftarrow w$ ;  $X_{min} \leftarrow X$ 
L11  if  $X_{min} = \emptyset$  then
L12     $(X_{min}, G[N]) \leftarrow D^*(f^\circ, w_{min}, G[N], \Delta)$ 
L13  if  $X_{min} = NOPATH$  then return  $NOPATH$ 
L14  if  $f_1(X_{min}) \leq K$  then return  $HIGHRANGE$ 
L15  if  $X_{max} = \emptyset$  then
L16     $(X_{max}, G[N]) \leftarrow D^*(f^\circ, w_{max}, G[N], \Delta)$ 
L17  if  $f_1(X_{max}) > K$  then return  $LOWRANGE$ 
L18  return  $(X_{max}, G[1..N])$ 
L19 end

```

Since  $D^*$  produces equivalent solutions to  $A^*$ ,  $CD^*$  produces equivalent results to  $CA^*$ . But is it significantly faster for re-planning? At the first invocation of  $CD^*$ , the algorithm constructs  $N$  graphs, one for each weight investigated. If the graph is to be modified with  $\Delta$  changes, a subsequent call to  $CD^*$  will again perform a binary search, incrementally updating each of the  $N$  graphs. Provided none of the *weights* change, all  $N$  of these incremental updates will be very fast. If the  $i$ -th weight changes, then  $D^*$  essentially plans from scratch for graphs  $i$  through  $N$ . The worst case occurs when the weight changes at position  $i = 2$  (it cannot change at  $i = 1$ ). In that case,  $D^*$  re-plans from scratch for  $N - 1$  graphs. This worst case is

comparable to calling  $CA^*$  on the modified graph. As shown in the experimental results section,  $CD^*$  re-plans its subproblems from scratch infrequently, so that  $CD^*$  is a significantly faster algorithm than  $CA^*$  for re-planning.

### 3. DISCUSSION

$CA^*$  is computationally efficient because it performs a binary search on the constraint dimension. Its complexity is  $NO(A^*)$ , rather than  $2^N O(A^*)$  for the equivalent exhaustive search.  $CD^*$  is even more efficient provided it avoids changing the weights often. Let  $p$  be the fraction of cases where a  $D^*$  graph is re-planned from scratch (e.g., due to a weight change) rather than updated. Then the complexity of  $CD^*$  is approximately  $N(pO(A^*) + (1-p)O(D^*))$ . If  $p$  is small, then  $CD^*$  is much faster at re-planning than  $CA^*$ , since  $D^*$  is much faster at re-planning than  $A^*$ . The parameter  $p$  is typically problem dependent.

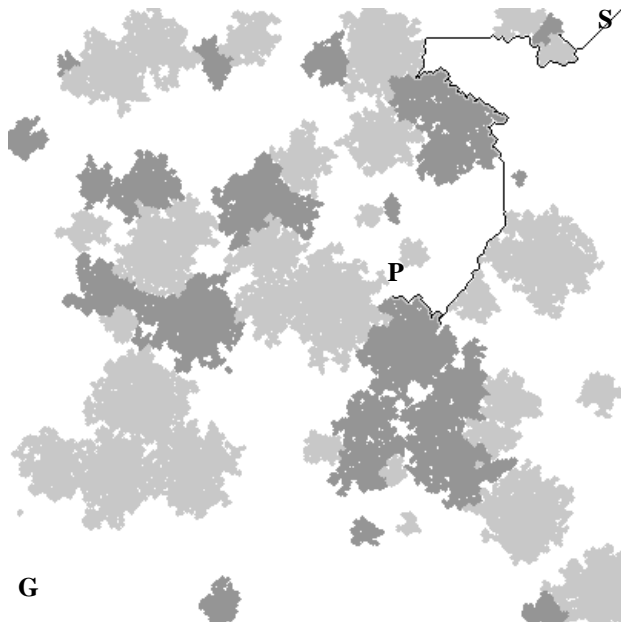
We can easily modify both  $CA^*$  and  $CD^*$  to terminate when the difference between the computed optimal solution  $f_0(X)$  and the exact optimal solution is less than some value,  $\epsilon$ , rather than terminate after a fixed number of iterations,  $N$ . This is done by evaluating  $f_0(X_{max}) - f_0(X_{min})$  at each iteration and stopping when the difference is less than  $\epsilon$ , or when the difference ceases to shrink (no solution).

To properly bound the optimal, constrained solution, the initial weights should be selected so that  $w_{min}$  yields an infeasible solution, if one exists, and  $w_{max}$  yields a feasible solution, if one exists. We do not have a rigorous rule for this selection; otherwise, we would incorporate it in the algorithm itself. Instead, we advocate a heuristic approach. We recommend  $w_{min} = 0$ , since zero is the lower bound for  $w$  in Equation 1. For  $w_{max}$ , it is important that the  $f_1^\circ$  term of Equation 1 dominate the  $f_0^\circ$  term, so that  $f_1^\circ$  is minimized. The selection  $w_{max} = \infty$  guarantees the proper solution if one exists, but large values of  $w$  require a commensurate increase in  $N$  (and corresponding increase in runtime and memory usage) in order to find a solution with the same weight resolution. So we recommend using the largest  $w_{max}$  possible that still yields required runtime performance. For the case where  $f_0^\circ$  and  $f_1^\circ$  are additive functions of positive costs, we recommend  $w_{max} = 10c_0/c_1$ , where  $c_0$  is the average additive weight for  $f_0^\circ$  and  $c_1$  is the average additive weight for  $f_1^\circ$ . If the optimal, constrained solution is improperly bounded, this condition is reported by both  $CA^*$  and  $CD^*$  at lines L14 and L17, and a new pair of bounds can be chosen.

### 4. ROUTE PLANNING PROBLEM

Consider the route planning problem described in the introduction, where the UGV plans a path to a goal that minimizes visibility (maximizes stealth) and arrives at the goal by a fixed time. To make the problem more realistic, the UGV knows about some of the obstacles in its environment (shown in light gray) but not about others (shown in

dark gray). The UGV is equipped with a sensor to detect the unknown obstacles. Whenever it detects an obstacle, it updates its map and re-plans the remainder of the path to the goal.



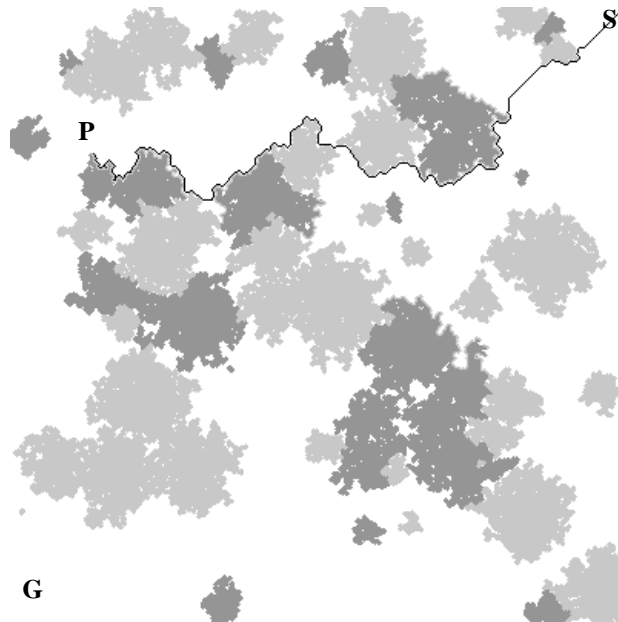
**Figure 1:** UGV traverse on first aborted attempt

Since the environment is not completely known, the UGV cannot guarantee that its traverse will not exhaust the time budget. Therefore, we establish the additional constraint that if the UGV determines, based on known information, that it cannot reach the goal without exhausting its time budget, it must return to the start. Under no circumstances may it be “caught in the middle” when the time expires. This is a realistic scenario since there could be known places to hide and wait at both the start and goal but not in between. Assume that  $T$  is the time remaining at any point in the traverse and  $T_{max}$  is maximum allowable time. The worst case occurs when the UGV travels all of the way to the goal and then discovers the goal is completely obstructed. Since the only paths with guaranteed time costs are those previously traversed, the UGV must then retreat to the start along its approach path. To allow for this contingency, we first call CD\* with time constraint  $K = T_{max}/2$  and begin traversing the planned route.

When the UGV’s sensor detects an obstacle, we update the map (i.e., edge costs in graphs) and invoke CD\* again. But we must also update the constraint, to account for the time consumed during execution. Let  $T_{trav}$  be the amount of time expended on the traverse so far. The constraint is updated to be  $K = (T_{max} - 2T_{trav})/2 = T_{max}/2 - T_{trav}$ , meaning that the maximum time available is the maximum allowable time minus twice the traversal cost so far (to permit a return), all divided by two (to account for worst case of goal blockage). A new plan is produced, and the process repeats until the goal is reached. At every point in a

traverse, the UGV follows a path that is optimal with respect to stealth and that satisfies the time constraint, taking into account all information known and detected in aggregate to that point. If the UGV determines it cannot reach the goal without exhausting the allowable time, it retreats to the start. It then ventures out again for another attempt, capitalizing on the new information it acquired from the previous traverse.

Figure 1 shows the first traverse taken by the UGV. The traverse was aborted (at point  $P$ ) and the UGV retreated, since it determined at  $P$  that it could not travel to the goal  $G$  and then retreat to  $S$  without violating the time constraint, given the field of known obstacles. Figure 2 shows a subsequent traverse, using the information acquired from previous traverses, exploring to the left of the center cluster of obstacles. Again, this traverse was aborted at point  $P$ . Figure 3 shows a later traverse, exploring to the opposite side, since the left side was found to be cluttered. This traverse was also aborted. Figure 4 shows the final attempt by the UGV that ended in success. With each attempt, the UGV discovered more of the obstacle clutter and progressively traded away stealth (i.e., perimeter following) for more direct paths to meet the time constraint.



**Figure 2:** UGV traverse exploring to the left side

## 5. EXPERIMENTAL RESULTS

Given the lack of other real-time re-planning algorithms available for this problem, we evaluated CD\* against CA\*. CA\* is not a re-planning algorithm, but it is an efficient *planning* algorithm. The optimal alternative to incremental re-planning is to plan from scratch each time new information arrives. This comparison is analogous to the earlier comparison of D\* to A\* (Stentz 1994). We generated environments of size 400 x 400 like the one in the previous section. Each environment consisted of 80 obstacles of maximum radial size 40 and randomly distributed.

The visibility cost of traversing open space was set to 10 times the cost of traversing an obstacle perimeter. Half the obstacles were unknown. The degree of the binary search ( $N$ ) was set to 8. The time constraint was set both *tight* (4X diagonal distance) and *loose* (8X diagonal). The performance was measured by running five experiments for each constraint and averaging the results. The algorithms were compared on a 600 MHz Pentium III, and the results are given in Table 1.

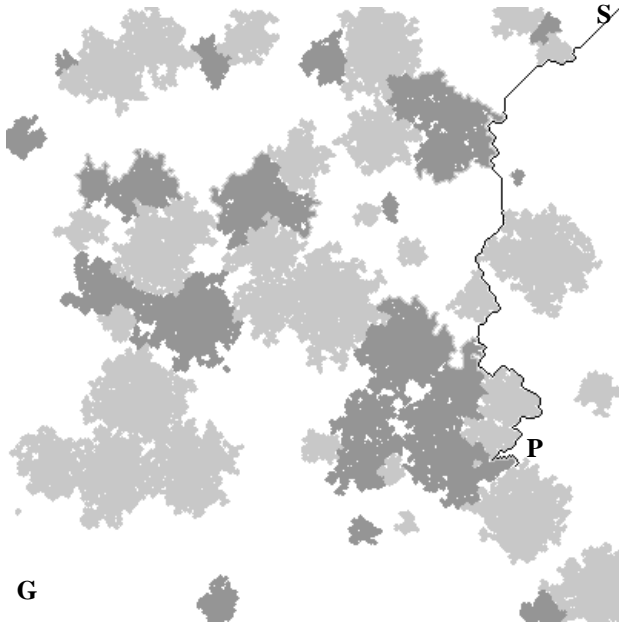


Figure 3: UGV traverse exploring to the right side

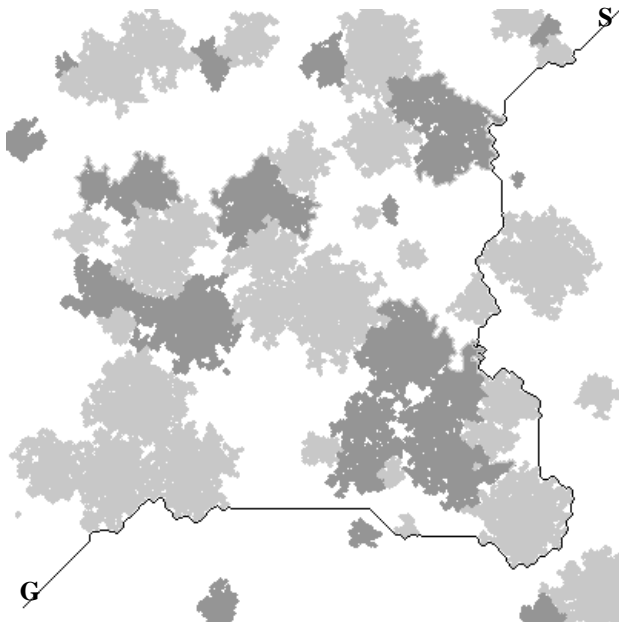


Figure 4: UGV traverse on final, successful attempt

The first data column lists the average number of attempts the UGV made to reach the goal, with all but the

Constraint	Attempts	No. of Steps	No. of Re-plans	CA* re-plan time (sec)	CD* re-plan time (sec)	CD* re-plan %
Tight	4.4	2654	372	3.51	0.35	7.7%
Loose	1.0	1290	367	3.27	0.045	0.7%

Table 1: Experimental results from simulation

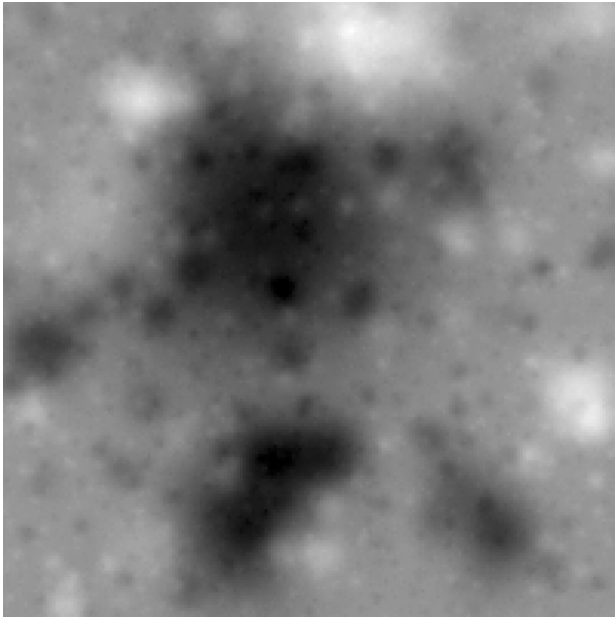
last ending in retreat. An attempt was aborted when the UGV determined that further progress could not be made while still guaranteeing a successful retreat in the event the goal was blocked. The second column lists the total number of steps (grid cells) in all attempts taken together. The third column lists the number of times the UGV detected new information, causing a reformulation of the constraint and a re-plan of the remaining path to the goal. The fourth column lists the average CPU time CA\* required to re-plan a path (once) during execution. The fifth column lists the average re-planning time for CD\*. The sixth column lists the percentage of times CD\* re-planned a graph from scratch rather than just updating it. Re-planning from scratch was required whenever one of the weights in the binary search changed.

From the experiments, CD\* is one to two orders of magnitude faster than CA\* for re-planning, even though CA\* by itself is a fairly fast algorithm. With a tight constraint, the UGV has less visibility “slack” to trade away to meet the constraint, and the detection of an unknown obstacle is more likely to abort an attempt. Furthermore, near the constraint boundary, the weights are more likely to change, causing CD\* to re-plan more of its graphs from scratch, thus slowing it down. With a loose constraint, the UGV has more visibility slack to trade away, enabling it to avoid unknown obstacles without risking time depletion. For the experiments conducted, all first attempts were successful. Furthermore, the UGV operates closer to the true unconstrained optimum with a loose constraint, and re-planning operations need only to incrementally update the optimal stealthy path.

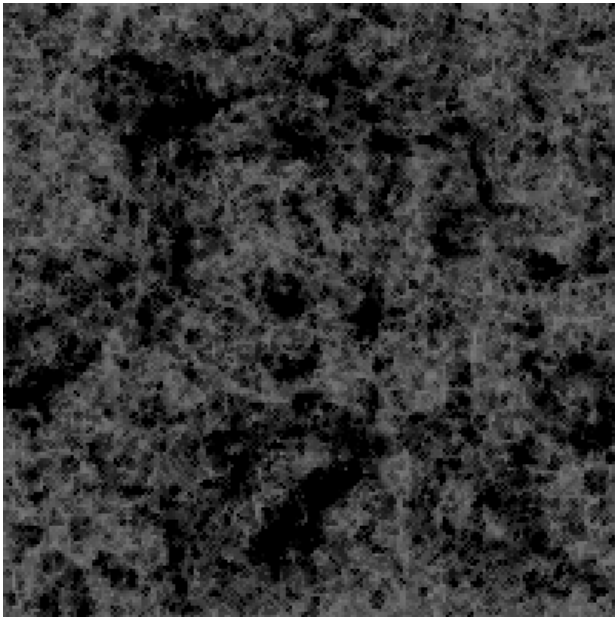
## 6. ILLUSTRATIVE SCENARIO

The following scenario illustrates the use of CD\* for a scout-like mission. Figure 5 shows a 200 x 200 cell terrain map with elevation encoded as gray scales. Black areas are low elevations (e.g., valleys or holes), and white areas are high elevations (e.g., ridges or hills). The terrain was generated using a fractal terrain simulator.

The geometry of the terrain determines the UGV’s mobility. Flat, level terrain can be traversed quickly; steep terrain more slowly; and large rocks, cliffs, and ravines, not at all. This last group comprises obstacles for the UGV. Figure 6 shows the terrain in Figure 5 processed for mobility. The white areas can be traversed quickly; the gray areas more slowly; and the black areas not at all. Note that some of the steep hills and depressions constitute obstacles in the mobility map.



**Figure 5:** Fractal terrain with gray scale elevation



**Figure 6:** Mobility cost map for fractal terrain

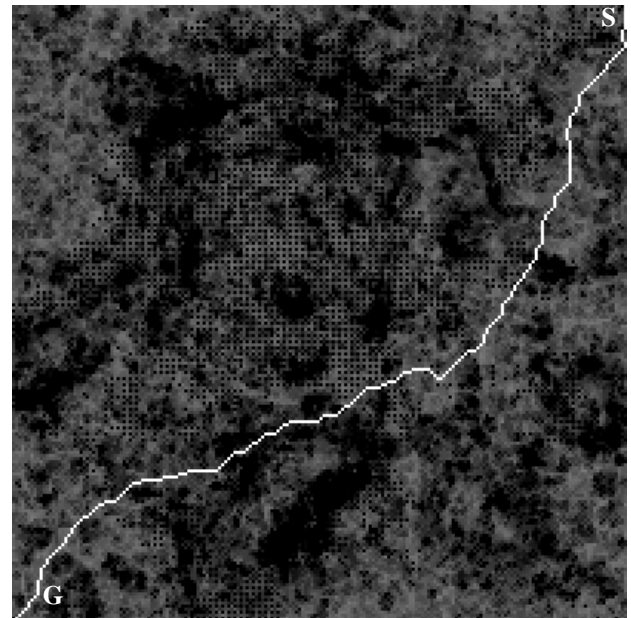
There is a known threat located in the center of the terrain. The portions of the terrain visible to this threat are shown as gray areas in Figure 7.

The UGV is tasked with departing a start position in the upper right corner at time  $T_1$  and arriving at a goal position in the lower left corner no later than time  $T_2$ . During the traverse, the UGV is instructed to avoid line of sight with the threat in the center (i.e., gray areas in Figure 7). If the UGV determines that it cannot reach  $G$  by time  $T_2$ , it is required to retreat to the start by time  $T_2$ .



**Figure 7:** Terrain areas visible to threat

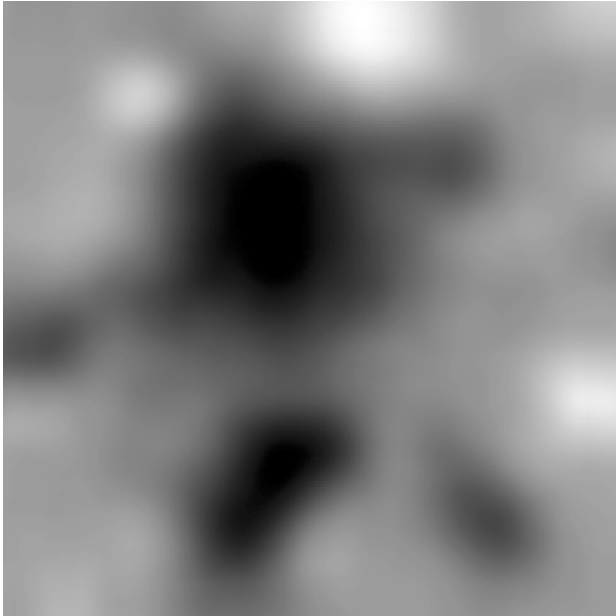
Figure 8 shows the optimal (i.e., stealthiest path) from  $S$  to  $G$  that satisfies the constraint. The UGV's path is shown as a white curve. The mobility and visibility maps are overlaid, with the gray scales denoting mobility cost and the checkered areas denoting visibility to the threat.



**Figure 8:** Stealthiest path meeting the time constraint

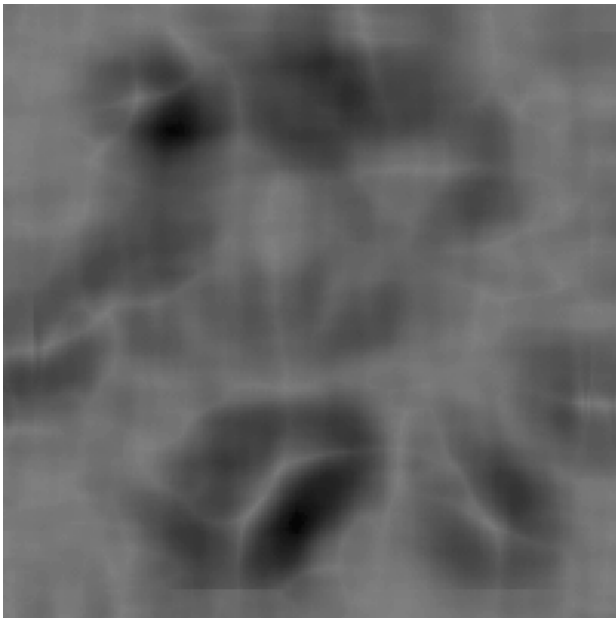
The UGV does not have perfect information about the terrain, however. Instead, it has a coarse map, shown in Figure 9, that is typical for satellite or overflight data. The map provides only approximate elevation data--it was cre-

ated by blurring the terrain in Figure 5 with a 10 x 10 cell filter.



**Figure 9:** Digital terrain elevation map used by UGV

Based on the approximate elevation data in Figure 9, the UGV computes an approximate mobility map (Figure 10) and approximate visibility map (Figure 11). Note that these maps constitute approximations to the true values, shown in Figure 6 and Figure 7 respectively.



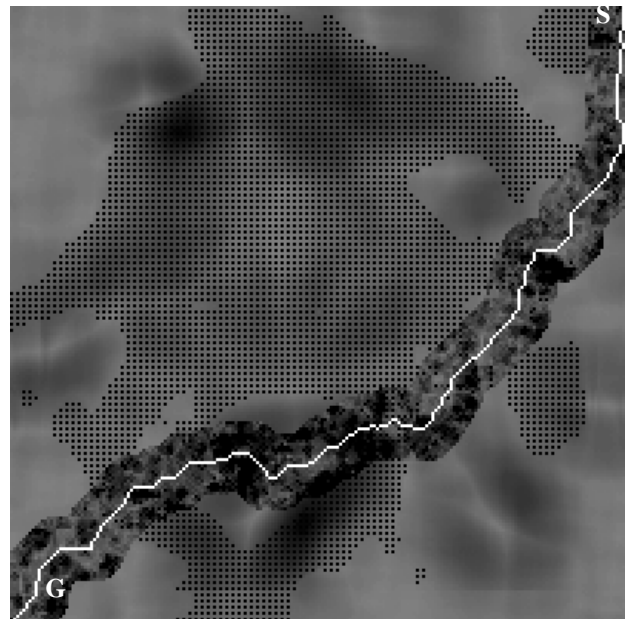
**Figure 10:** Approximate mobility cost map

The UGV is equipped with a radial sensor for detecting the terrain elevation as it drives. It starts at S and plans an initial route that maximizes stealth and meets the time constraint. It then begins to drive the route, sensing the ter-

rain as it goes. For each image collected, it updates its digital terrain elevation map (Figure 9). From the updated terrain map, it also updates its mobility (Figure 10) and visibility (Figure 11) maps.



**Figure 11:** Approximate visibility map



**Figure 12:** UGV's actual traverse

Using CD\*, the UGV re-plans the remainder of its path, based on the updated maps. Its sensor may uncover previously unknown obstacles, forcing the UGV to take a longer path, and thus to trade away some stealthiness in order to meet the time constraint. The sensor may also discover unexpected hiding places, or eliminate expected ones, forcing the UGV to adjust its path accordingly to

minimize visibility. This process repeats until the UGV reaches  $G$ . If the UGV determines it cannot arrive at  $G$  and meet the time constraint, it retreats to  $S$ .

Figure 12 shows the traverse taken by the UGV. The band of data surrounding the white curve represents the updated mobility and visibility costs for the terrain sensed by the UGV. Note that it is higher resolution than the initial data. The traverse is different than the initial path (Figure 8), since the sensed terrain forced the UGV to re-plan as it went. At every point in the traverse, the UGV is following the stealthiest path that meets the time constraint.

## 7. CONCLUSIONS

This paper describes an algorithm, CD\*, for real-time re-planning of optimal paths that satisfy a global constraint. The algorithm is fast enough to compute new paths for every new piece of sensor information detected by a UGV, and it can be integrated directly into a UGV's controller. We are presently working to extend CD\* to handle multiple constraints and expect that each new constraint will add a factor of  $N$  in run-time complexity, rather than  $2^N$  for the exhaustive search case.

## ACKNOWLEDGMENTS

This work was sponsored in part by the U.S. Army Research Laboratory, under contract "Robotics Collaborative Technology Alliance" (contract number DAAD19-01-2-0012). The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies or endorsements of the U.S. Government.

## REFERENCES

- Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. San Francisco, California: Freeman.
- Hassin, R. 1992. Approximation schemes for restricted shortest path problem. *Mathematics of Operations Research* 17(1):36-42.
- Jaffe, J. M. 1984. Algorithms for Finding Paths with Multiple Constraints. *Networks* 14:95-116.
- Korkmaz, T., Krunz, M., and Tragoudas, S. 2000. An Efficient Algorithm for Finding a Path Subject to Two Additive Constraints. In *Proceedings of the ACM SIGMETRICS 2000 Conference*, 318-327, Santa Clara, CA.
- Korkmaz, T., and Krunz, M. 2001. Multi-Constrained Optimal Path Selection. In *Proceedings of the IEEE INFOCOM 2001 Conference*, 834-843, Anchorage, Alaska.
- Logan, B., and Alechina, N. 1998. A\* with Bounded Costs. In *Proceedings of the 15th National Conference on Artificial Intelligence*, 444-449. AAAI Press.
- Lui, G., and Ramakrishnan, K. G. 2001. A\* Prune: An Algorithm for Finding K Shortest Paths Subject to Multiple Constraints. In *Proceedings of the IEEE INFOCOM 2001 Conference*, Anchorage, Alaska.
- Stentz, A. 1994. Optimal and Efficient Path Planning for Partially-Known Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, California.
- Stentz, A. 1995. The Focussed D\* Algorithm for Real-Time Replanning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1652-1659, Montreal, Canada.
- Stentz, A. 2002. CD\*: A Real-Time Resolution Optimal Re-Planner for Globally Constrained Problems. In *Proceedings of AAAI 2002*, Edmonton, Canada.