

CD*: A Real-time Resolution Optimal Re-planner for Globally Constrained Problems

Anthony Stentz

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

Abstract

Many problems in robotics and AI, such as the find-path problem, call for optimal solutions that satisfy global constraints. The problem is complicated when the cost information is unknown, uncertain, or changing during execution of the solution. Such problems call for efficient re-planning during execution to account for the new information acquired. This paper presents a novel real-time algorithm, Constrained D* (CD*), that re-plans resolution optimal solutions subject to a global constraint. CD* performs a binary search on a weight parameter that sets the balance between the optimality and feasibility cost metrics. In each stage of the search, CD* uses Dynamic A* (D*) to update the weight selection for that stage. On average, CD* updates a feasible and resolution optimal plan in less than a second, enabling it to be used in a real-time robot controller. Results are presented for simulated problems. To the author's knowledge, CD* is the fastest algorithm to solve this class of problems.

Introduction

Many problems in robotics and AI call for optimal solutions, for instance, the find-path problem for manipulators and mobile robots, automated component assembly, job shop scheduling, reinforcement learning, and others. The optimization problem can be cast as minimizing or maximizing some objective function, such as distance travelled, time elapsed, energy consumed, jobs processed, information gained, and error between predicted and observed data. Often, the set of feasible solutions is limited by one or more constraints, which can be local or global. A local constraint can be evaluated at a single step in the solution, such as the constraint that a mobile robot must avoid all obstacles in the find-path problem. A global constraint applies to the entire solution, or at least a large portion of it, such as the constraint that a mobile robot must reach its goal before exhausting its battery.

For example, consider a variant of the find-path problem. A mobile robot is tasked with finding the stealthiest obstacle-free path to the goal. A "stealthy path" is one that moves along the perimeter of obstacles to minimize visibility. Additionally, the robot must reach the goal before exhausting its battery. To optimize its objective function,

the robot favors paths that hop from obstacle to obstacle and avoid cutting across open areas. But these longer paths are precisely the ones that are likely to violate the global energy constraint.

Assuming a feasible and optimal solution can be produced, it may need to be modified as the "plan" is executed. In the example given, the mobile robot 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 robot waits for the replanner 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 introduces the CD* (Constrained D*) algorithm, 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, prove its correctness, and measure its speed through experiments in simulation.

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 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 energy 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 (see next section), 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.

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 robot may discover that a door is unexpectedly closed, thereby requiring the robot to re-plan the remaining path to the goal. The algorithm 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° 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 Δ represents changes to G (e.g., costs or connectivity). Initially, G is set to \emptyset and Δ is set to the entire graph itself. Subsequent calls to D* on G with changes Δ 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 Δ 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.

Proofs of Correctness

Like D*, CD* also operates on a graph, $G = (V, E)$, consisting of vertices V and edges E . We can think of the vertices as states and the edges as *actions* that transition from one state to another. For vertices (states) we use lower case letters, such as x and y . For edges (actions) we use the function $c(Y, X)$, which returns the action costs of moving across the edge from x to y . Capital letters refer to sequences of states $X = \{x_0, x_1, x_2, \dots, x_i\}$. X_i refers to the subsequence $\{x_0, x_1, \dots, x_i\}$.

A *path function* f° is defined to be a function of a sequence X . The function can be specified recursively as $f(X_i) = f(c(X_i, X_{i-1}), f(X_{i-1}))$ and $f(X_0) = 0$. Consider two sequences, X and Y , that share the first state. A path function f° is defined to be a *dynamic programming (DP) function* if $f^*(X_i) = f(c(X_i, X_{i-1}), f^*(X_{i-1}))$ and if $f(X_i) \geq f(X_{i-1})$ for all X and i . The first condition states that the optimal path for sequence X_i is strictly a function of the optimal sequence to X_{i-1} and the connecting action. The second condition states that f° is monotonic. In Equation 1, we define f° , f_0° , and f_1° to be path functions; f° to be a DP function; and w to be a non-negative weight.

Consider the following definitions that will be used in the theorems below:

Definition 1: Let X_w be the sequence that minimizes $f(X, w)$ in Equation 1. If multiple minimizing sequences exist, X_w is defined to be a sequence that minimizes $f_1(X)$.

Definition 2: Let X_u be the sequence that minimizes $f(X, u)$ in Equation 1. If multiple minimizing sequences exist, X_u is defined to be a sequence that minimizes $f_1(X)$.

The following theorem proves the necessary condition for the binary search to operate, namely that increasing the weight w in Equation 1 drives down the constraint value and drives up the objective function value.

Theorem 1: If $u > w$, then $f_0(X_u) \geq f_0(X_w)$ and $f_1(X_u) \leq f_1(X_w)$.

Proof: we prove the above pair of relationals by disproving the other three possible cases. Case 1: $f_0(X_u) \geq f_0(X_w)$ and $f_1(X_u) > f_1(X_w)$. Then $f_0(X_w) + uf_1(X_w) < f_0(X_u) + uf_1(X_u)$ and Definition 2 is violated. Case 2: $f_0(X_u) < f_0(X_w)$ and $f_1(X_u) \leq f_1(X_w)$. Then $f_0(X_u) + wf_1(X_u) < f_0(X_w) + wf_1(X_w)$ and Definition 1 is violated. Case 3: $f_0(X_u) < f_0(X_w)$ and $f_1(X_u) > f_1(X_w)$. For X_u to be the optimal sequence for $f(X, u)$, then $f_0(X_u) + uf_1(X_u) \leq f_0(X_w) + uf_1(X_w)$. This equation can be rewritten as $f_0(X_w) - f_0(X_u) \geq u(f_1(X_u) - f_1(X_w))$. Since $u > w$, then $u(f_1(X_u) - f_1(X_w)) > w(f_1(X_u) - f_1(X_w))$. Substituting the latter equation into the former, we have $f_0(X_w) - f_0(X_u) > w(f_1(X_u) - f_1(X_w))$. This equation can be rewritten as $f_0(X_u) + wf_1(X_u) < f_0(X_w) + wf_1(X_w)$, which violates Definition 1. The remaining case is $f_0(X_u) \geq f_0(X_w)$ and $f_1(X_u) \leq f_1(X_w)$. QED.

We now prove the correctness of CA*.

Theorem 2: Let W be the set of weights w_i that evenly divides the range $[w_{min}, w_{max}]$ into $2^{N-1} + 1$ weights inclusive (provided $N > 0$ and $w_{min} < w_{max}$). Let X_{wo} be a sequence with the smallest weight w_o such that X_{wo} minimizes $f_0(\circ)$ and X_{wo} satisfies $f_1(\circ)$. CA* returns either 1) a sequence X , such that $f_0(X) \leq f_0(X_{w_i})$ for all w_i for which $f_1(X_{w_i}) \leq K$, if $w_{min} < w_o \leq w_{max}$, 2) *LOWRANGE*, if $w_o > w_{max}$ or X_{wo} does not exist, 3) *HIGHRANGE*, if $w_o \leq w_{min}$, or 4) *NOPATH*, if no path of any measure to the goal exists.

Proof: Let w_k be the smallest threshold weight such that $f_1(X_{w_k}) \leq K$. From Theorem 1, increasing the weight w non-decreases $f_0(\circ)$ and non-increases $f_1(\circ)$. Therefore, all weights $w < w_k$ are infeasible. Again from Theorem 1, all weights $w \geq w_k$ are feasible, and the smallest such weight has the minimal $f_0(\circ)$; therefore, $w_k = w_o$ and $X_{w_k} = X_{w_o}$. CA* evaluates a discrete set of weights. From Theorem 1, it follows directly that the smallest w_i such that $w_i \geq w_k$ minimizes $f_0(\circ)$ and satisfies $f_1(\circ)$ across all w_i in W . CA* performs a binary search on w to find the smallest such w_i .

The search converges on w_k and then selects the smallest w_i greater than or equal to w_k after exiting the loop. At each iteration through the loop (L3 through L10), CA* divides the interval between the upper and lower bounds on w_k in half and establishes a new upper or lower bound. The sequences and the weights for both bounds are recorded. The proper lower bound (w_{min}) on w_k must be infeasible, since by Theorem 1 the corresponding sequence must have a larger $f_1(\circ)$ value than $f_1(X_{w_k}) = K$. Similarly, the proper upper bound (w_{max}) on w_k must be feasible. Thus, the appropriate bound is adjusted by examining the feasibility of w at line L7.

After exiting the loop, the proper weight is selected. If the original input weights bound w_k (case 1), then the updated weights preserve this property: $w_{min} < w_k \leq w_{max}$. Furthermore, w_{min} and w_{max} represent consecutive weights in the sequence w_i for $i = 1$ to $2^{N-1} + 1$. Therefore, the smallest w_i such that $w_i \geq w_k$ is w_{max} (L18). If the original weight range is below w_k (i.e., case 2: $w_k > w_{max}$), then all weights in the range yield infeasible sequences (Theorem 1). Only w_{min} is updated (L10) in each iteration of the loop, and w_{max} retains its original value yielding an infeasible sequence. This condition is detected at line L17 and *LOWRANGE* is returned. Similarly, if the original weight range is above w_k (i.e., case 3: $w_k \leq w_{min}$), then all weights in the range yield feasible sequences (Theorem 1). In this case, w_{min} retains its original value, and *HIGHRANGE* is reported at line L14.

Finally, there is no path of any measure, inside or outside of the weight range, if the first call to A*(\circ) returns no path (L13 if $N = 1$ and L6 if $N > 1$). This is case 4. Changing the weight on a term in the objective function $f(\circ)$ does not affect whether or not a path through the graph exists. QED.

We now prove the correctness of CD*.

Theorem 3: If $G' = G + \Delta$, $X = CA^*(f(\circ), w_{min}, w_{max}, N, G')$, and $(Y, G[1 \dots N]) = CD^*(f(\circ), w_{min}, w_{max}, N, G[1 \dots N], \Delta)$, then $f_0(X) = f_0(Y)$ and $f_1(X) = f_1(Y)$.

Proof: There are two differences between CA* and CD*: 1) CA* calls A* and CD* calls D*, and 2) CA* operates on a single input graph and CD* operates on an array of graphs. Since all elements of the array are initialized to G , D* operates on G in every stage of the binary search. Since $A^*(f(\circ), w, G')$ and $D^*(f(\circ), w, G, \Delta)$ produce equivalent results (i.e., $f(\circ)$ values are the same), CA* and CD* produce equivalent results. QED.

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, ϵ , 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 ϵ , 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.

Although we require $f(\circ)$ to be a DP function so that we can use both A* and D*, note that there is no requirement that $f_0(\circ)$ and $f_1(\circ)$ be DP functions. This means that our formulation is broader than that used in QoS, where path functions are strictly additive and the edges of the graph non-negative. For example, assume we would like to compute the lowest cost path through a network of both positive and negative weights, such that the path has no more than M segments. Therefore, we would like to minimize

$$\text{Equation 2: } f_0(X_M) = \sum_{i=1}^M c_i(X_i, X_{i-1})$$

where $c_i(\circ)$ are the edge costs. Assuming there is at least one negative $c_i(\circ)$ in the graph, define C to be the absolute value of the most negative edge in the graph. The constraint can be defined as $f_1(X_M) = MC$ with the threshold $K = MC$. Therefore,

$$\text{Equation 3: } f(X_M) = \sum_{i=1}^M c_i + wMC = \sum_{i=1}^M (c_i + wC).$$

If $w \geq 1$, then each term in the summation is non-negative. Thus, $f(\circ)$ is monotonically non-decreasing and is a DP function, even though $f_0(\circ)$ is not. Both CA* and CD* can be used to solve this problem by setting $w_{min} = 1$.

Route Planning Example

Consider the route planning example described in the introduction, where the robot plans a path to a goal that minimizes visibility (maximizes stealth) without exhausting its battery. To make the problem more realistic, the robot knows about some of the obstacles in its environment (shown in light gray) but not about others (shown in dark gray). The robot is equipped with a contact 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.

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

When the robot’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 energy expended during execution. Let E_{trav} be the amount of energy expended on the traverse so far. The constraint is updated to be $K = (E_{max} - 2E_{trav})/2 = E_{max}/2 - E_{trav}$, meaning that the maximum energy available is the battery capacity 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 robot follows a path that is optimal with respect to stealth and that satisfies the energy constraint, taking into account all information known and detected in aggregate to that point. If the robot determines it cannot reach the goal without exhausting its battery, it retreats to the start and recharges. 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 robot. The traverse was aborted (at point P) and the robot retreated, since it determined at P that it could not travel to the goal G and then retreat to S without violating the energy 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 robot that ended in success. With each attempt, the robot discovered more of the obstacle clutter and progressively traded away stealth (i.e., perimeter following) for more direct paths to meet the energy constraint.

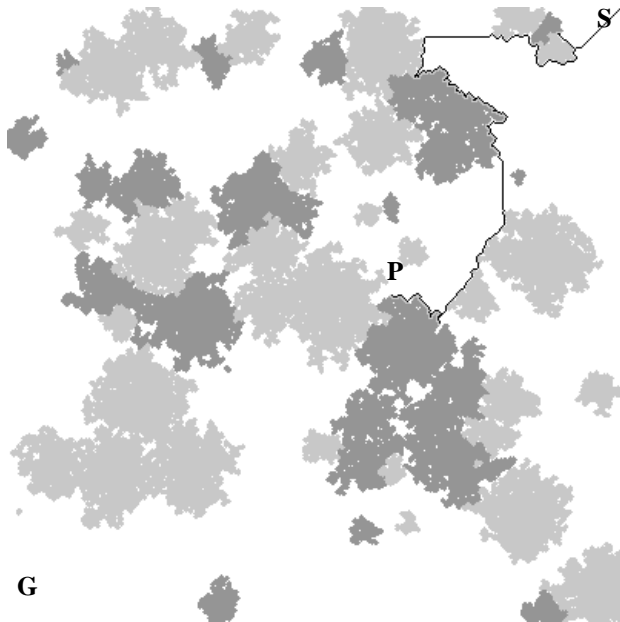


Figure 1: Robot traverse on first aborted attempt

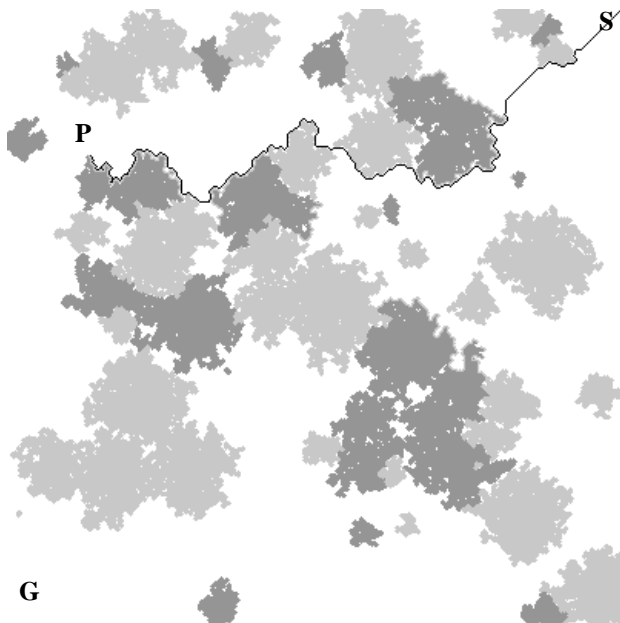


Figure 2: Robot traverse exploring to the left side

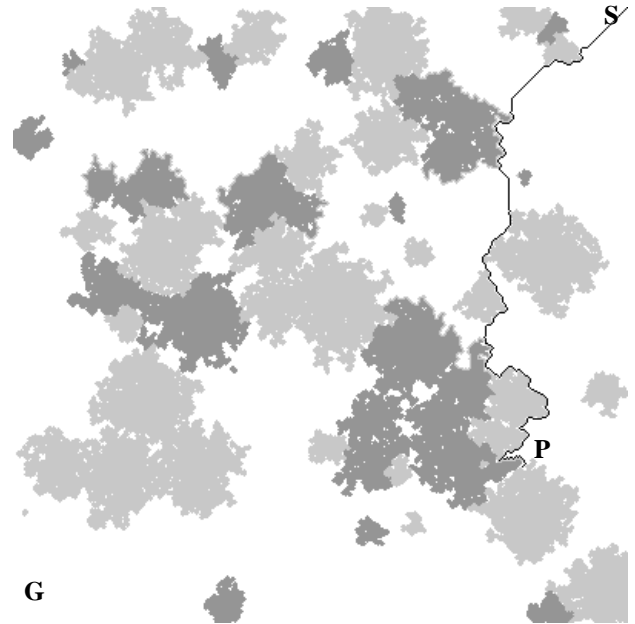


Figure 3: Robot traverse exploring to the right side

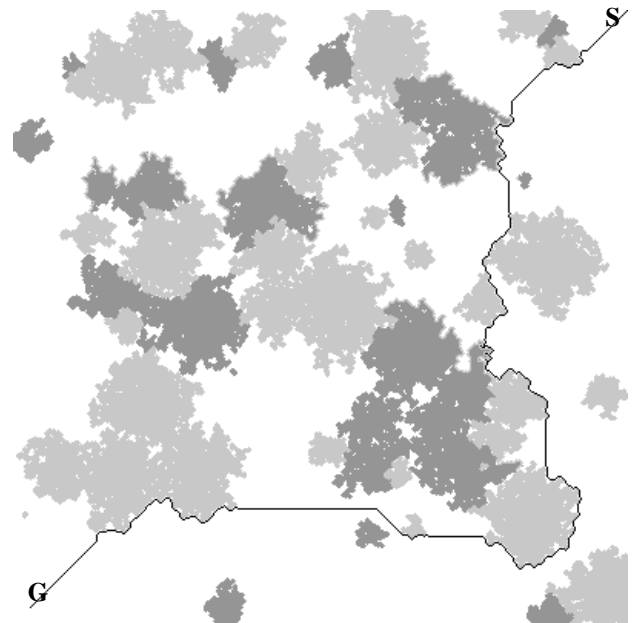


Figure 4: Robot traverse on final, successful attempt

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

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 energy 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.

The first data column lists the average number of attempts the robot made to reach the goal, with all but the last ending in retreat. An attempt was aborted when the robot 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 robot 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 robot 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 robot has more visibility slack to trade away, enabling it to avoid unknown obstacles without risking energy depletion. For the experiments conducted, all first attempts were successful. Furthermore, the robot operates closer to the true unconstrained optimum with a loose constraint, and re-planning operations need only to incrementally update the optimal stealthy path.

Summary and Future Work

This paper presents a novel 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 robot, and can be integrated directly into a robot’s controller. Although the examples and experiments in this paper were motivated by robot route planning problems, the formulation is very general, and we expect CD* to be useful for a wide variety of AI applications, in the same way that both A* and D* are. 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.