# Planning Multi-Goal Tours for Robot Arms

Mitul Saha[(1)] Gildardo Sánchez-Ante[(2)] Jean-Claude Latombe[(1)]

(1) Computer Science Department, Stanford University, Stanford, CA, USA

(2) Computer Science Department, ITESM-Campus Guadalajara, Guadalajara, México

## Abstract

*This paper considers the multi-goal motion planning problem. A robot arm must reach several input goal configurations in some sequence, but this sequence is not given. Instead, the robot's planner must compute an optimal or near-optimal path through the goals. This problem occurs, for instance, in spot-welding, inspection, and measurement tasks. It combines two sub-problems – the shortest-path and the traveling-salesman problems – that are both known to be computationally hard. This paper describes a heuristic algorithm that operates under the assumption that the number of goals is relatively small (a few dozen at most) and the computational cost of finding a good path between two goals dominates that of finding a good tour in a graph with edges of given costs. Although the algorithm computes a quadratic number of goal-to-goal paths in the worst case, it is much faster in practice. This paper gives a thorough experimental analysis of the planner.*

## 1 Introduction

Consider a robot that must reach several input goal configurations in some sequence, but this sequence is not given. Instead, the robot's planner must determine the sequence that yields a path of minimal length through the goals (e.g., a path requiring minimal execution time). The robot stops at each goal, where its end-effector performs some operation. The time taken by this operation is independent of the goal ordering. This multi-goal planning (MGP) problem occurs frequently in practice, e.g., in spot-welding, inspection, and measurement tasks. Figure 1 shows a typical spot-welding workstation in an automotive body shop. The robot arm must bring the welding gun at successive positions where it performs spot-welding operations to assemble body parts.

The above multi-goal problem combines two computationally hard problems: the shortest-path and the traveling-salesman (TSP) problems. The first problem must be solved to determine optimal collision-free paths between pairs of goals and their lengths. The second problem refers to the computation of an optimal tour in a graph whose edges are weighted by known goal-to-goal path lengths. Even the basic Euclidean shortest-path problem among polyhedral obstacles is NP-hard [3]. Finding a shortest tour through points in the Euclidean plane is also NP-hard [10, 12]. However, for both problems there exist efficient approximation and/or heuristic algorithms to compute sub-optimal, but still satisfactory solutions.

Figure 1: Spot-welding station

In this paper, we assume that we are given two functions:

- PATH – Given two robot configurations, this function returns a "good" collision-free path between them if they can be connected without collision. It indicates that no path exists otherwise.
- TOUR – Given a collection of points with given distances between all pairs, this function returns a "good" tour through these points that start and end at the same point.

A naive approach to compute a multi-goal path is to apply PATH to every pair of goals – hence, a quadratic number of times – before running TOUR once. However, in robotics problems, the number of goals typically ranges between 5 and 50, which is considered small for most approximate TSP algorithms. In contrast, robot paths must be computed in complex configuration spaces, so that the running time of PATH is usually much greater than that of TOUR. Thus, we wish to use PATH sparingly, even if this requires running TOUR (or a portion of it) more frequently. We do this by setting the length of a path between two goals to a lower bound until PATH is applied to this pair of goals. Each call of TOUR applies to a graph of goals in which some edge lengths are exact (i.e., have been computed by PATH) and some edge lengths are lower bounds. After each run of TOUR, PATH is invoked to obtain the actual path length between selected pairs of goals. In the worst-case, PATH is applied to all pairs of goals. But, in practice, our algorithm avoids many calls to PATH and returns a tour in much less time than the naive approach. We show that when TOUR is based on the algorithm given in [5, 12] that "doubles" the minimum spanning tree of the goals, our planning algorithm computes a tour whose length is within a given factor $2\alpha$, where $\alpha \geq 1$ is an input real number, of the tour computed by the naive approach.

Section 2 relates our work to previous work on multi-goal planning. Section 3 gives a precise statement of the multi-goal planning problem studied in this paper. Section 4 describes our planning algorithm. Section 5 describes the implemented function PATH. Section 6 analyzes the experimental performance of the planner on several examples.

## 2 Relation to Previous Work

(1) In [14] the multi-goal planning problem is considered in the context of coordinate measuring machines (CMM's). Each goal defines a position of the CMM's probe. The proposed algorithm uses a multi-query PRM (Probabilistic Road-Map) path planner [9] to compute goal-to-goal paths. A roadmap is pre-computed over the probe's free space. Its nodes are the goal configurations, plus additional configurations picked at random until all the goals are in the same connected component. A search algorithm then extracts the shortest path (in the roadmap) between every two goals. This yields a reduced, complete graph whose nodes are exactly the goal configurations and in which each edge is labelled by the length of the path extracted between the two goals it connects. A TSP algorithm computes a near-optimal tour from this graph. As the lengths labelling the edges satisfy the triangular inequality, polynomial-time algorithms are available to compute a tour within a constant ratio of the optimal tour [5, 11].

This method is based on a set of coherent choices. The CMM problems considered in [14] are formulated in quasi-planar configuration space (where a configuration is the position of the tip of the measuring probe). In the two examples given in [14], the roadmaps only contain 100 nodes (including the goals). Thus, pre-computing them does not take much time. Moreover, once a roadmap is available, the search of a shortest path between every two goals is very fast. This justifies the fact that all goal-to-goal paths are computed before a tour is extracted.

Our multi-goal planner is based on different assumptions. In many applications (like spot welding), the cost of pre-computing a roadmap is high, due to both the high dimensionality of the configuration space and the geometric complexity of the obstacles and the robot. It may then take a large number of goal-to-goal planning operations – often several hundreds – for a multi-query PRM planner to be more efficient in amortized time than a single-query PRM planner that does not pre-compute a roadmap [8, 13]. For this reason, the PATH function in our multi-goal planning algorithm is based on a single-query PRM planner (more details will be given in Section 5). But each evaluation of PATH is expedited by re-using the roadmap-construction work done at previous evaluations. Furthermore, in most problems, whether a multi- or single-query PRM planner is used, the average amortized computational cost per goal-to-goal planning operation is large relative to that of running a TSP algorithm. Hence, our algorithm strives to compute a tour without computing a path between all pairs of goals. This is consistent with the fact that our function PATH is based on a single-query PRM planner that does not pay the up-front cost of pre-computing a roadmap.

(2) The multi-goal planner described in [15] tries to avoid computing a collision-free path between every pair of goals. But, unlike ours, it neither uses lower bounds on path lengths, nor the information contained in intermediate tours computed by the TSP algorithm to guide the selection of the pairs of goals between which paths are computed. Instead, this planner loops on the following operations: select pairs of goals using a technique like random selection or nearest-pair selection, compute a collision-free path between each pair, run the TSP algorithm. Once the TSP algorithm has returned a first complete tour, the planner can stop at any time, with each new computed tour being at least as good as the previous one. But, if the planner is stopped before all goal-to-goal paths have been generated, then there is no formal guaran-

tee on the goodness of the last computed tour. In contrast, our algorithm provides such a guarantee ($2\alpha$ approximation factor). The planner in [15] uses a best-first search technique to generate goal-to-goal paths; this technique is inherently limited to searching low-dimensional configuration spaces, which could restrict the range of problems it can handle.

(3) The multi-goal planner in [6] is aimed at generating inspection tours, for example finding cracks in large structures. The robot, which is modeled as a point in a polygonal free space $F$, must go to successive goal positions such that it can see every point in the boundary of $F$ from at least one of these positions. This problem is also known as the watchman-route problem. Unlike in our problem, the goals are not given. In [6] they are computed using the approximate randomized "art-gallery" algorithm given in [7]. Next, the visibility graph of these goals and the vertices of $F$'s boundary is computed and the shortest path between every two goals is computed [11]. An approximation TSP algorithm extracts a tour that is at most twice as long as the optimal tour. An extension to 3-D workspace is discussed in [6].

(4) Other variants of MGP problems, with no or few collision-avoidance constraints, have been addressed using general optimization techniques such as simulated annealing [4] or genetic algorithms [2].

## 3 Problem Formulation

We let $C$ denote the configuration space of a robot and $F \subseteq C$ its collision-free subset. We let $\ell$ stand for the selected measure that maps any given path $\tau$ in $C$ to its length $\ell(\tau)$.

Let $g_0, g_1, ..., g_r$ be $r+1$ input configurations in $F$ called *goals*, with $g_0$ being also both the *start* and *end* configuration of the robot. These $r+1$ configurations are all distinct.

Any path in $F$ joining two distinct goals $g_i$ and $g_j$ is called a *goal-to-goal* path. Any loop path in $F$ starting and ending at $g_0$, and passing through every goal $g_i$, $i = 1$ to $r$, is called a *multi-goal path*. So, a multi-goal path $\tau$ is a sequence of $r+1$ goal-to-goal paths. The length of $\tau$ is assumed to be additive, i.e., it is the sum of the lengths (measured by $\ell$) of the goal-to-goal paths it contains. This is reasonable since in most applications the robot stops at each goal; so, the goals divide the multi-goal path into "independent" pieces. The *multi-goal planning problem* is to find the shortest multi-goal path, or an approximation of it.

---

**Algorithm** TOUR($G$)
1. $T \leftarrow$ MIN-SPANNING-TREE($G$)
2. Return PREORDER-WALK($T$)

---

Figure 2: TOUR algorithm

The function PATH defined as follows is given. For any two goals $g_i$ and $g_j$, PATH($g_i, g_j$) returns a path in $F$ joining $g_i$ and $g_j$, if these configurations lie in the same component of $F$. For general robot arms, no efficient algorithm is available to compute a goal-to-goal path that is guaranteed to be within some approximation factor of the shortest path. So, PATH is by necessity a heuristic algorithm and we assume that the goal-to-goal paths it returns are "good" enough. In our implementation, PATH is a single-query PRM planner with an optimization post processing step (Section 5). Without loss of generality, we assume that the three paths returned by PATH between any three goals satisfy the triangular inequality. (This assumption can always be trivially enforced, if needed.)

We define the *goal graph* to be the complete non-directed graph $G = (V, X, c)$ with weighted edges. $V = \{g_0, g_1, ..., g_r\}$ is the set of vertices and $X = \{\{g_i, g_j\} \mid i, j \in \{0, 1, ..., r\}\}$ the set of edges. Each edge $\{g_i, g_j\}$ is weighted by a positive real number $c(g_i, g_j)$, the edge's *cost*. A TOUR of $G$ is any list $\pi = (g_0, g_{i_1}, ..., g_{i_r}, g_0)$ of all vertices starting and ending at $g_0$. Hence, $< i_1, ..., i_r >$ is a permutation of $\{1, ..., r\}$. The *cost* $c(\pi)$ of $\pi$ is the sum of the costs of the edges along the tour, hence:

$$c(\pi) = \sum_{k=1}^{k=r+1} c(g_{i_{k-1}}, g_{i_k}) \quad \text{with} \quad i_0 = i_{r+1} = 0.$$

Similarly, we define the cost $c(T)$ of a spanning tree $T$ of $G$ as the sum of the costs of the edges contained in $T$. A *minimum* tour (resp., spanning tree) is one that has minimum cost over all possible tours (resp., spanning trees).

The function TOUR defined in Figure 2 computes a tour of $G$ by "doubling" the minimum spanning tree of $G$ [5, 12]. This TSP algorithm was also used in the multi-goal planners described in [6, 14] In this function, MIN-SPANNING-TREE($G$) evaluates to a minimum spanning tree $T$ of $G$ rooted at $g_0$ and PREORDER-WALK($T$) returns the list of vertices of $G$ visited in a preorder walk of $T$. (A preorder walk of a tree recursively visits every vertex in the tree, starting at the root, listing a vertex when it is first encountered.) The cost of $T$ is no greater than that of the minimum

**Algorithm** NAIVE-MGP($G$)
1. Set $G$ to $(V, X, c)$
2. $V \leftarrow \{g_0, g_1, ..., g_r\}$
3. $X = \{\{g_i, g_j\} \,|\, i, j \in \{0, 1, ..., r\}\}$
4. For every $e = \{g_i, g_j\} \in E$ set $c(e)$ to $\ell(\text{PATH}(g_i, g_j))$
5. Return multi-goal path determined by TOUR($G$)

Figure 3: Naive multi-goal algorithm

tour. Moreover, when edge costs satisfy the triangular inequality, the cost of the tour PREORDER-WALK($T$) is no greater than twice that of $T$, hence within approximation factor 2 of the cost of the minimum tour [5]. Since $G$ is complete, both MIN-SPANNING-TREE and TOUR run in time $\Theta(r^2)$.

Consider the NAIVE-MGP algorithm in Figure 3. To compute a multi-goal path, it first weighs every edge $\{g_i, g_j\}$ of the goal graph $G$ by the length of the goal-to-goal path between $g_i$ and $g_j$ generated by PATH. (If PATH ever fails to find a path between two goals, then NAIVE-MGP returns *failure*, since no solution can possibly be found.) Then, it evaluates TOUR($G$) once. The concatenation of the goal-to-goal paths between every two successive goals in the tour returned by TOUR($G$) is the multi-goal path computed by NAIVE-MGP. Let $\mathcal{L}_N$ stand for the length of this path.

NAIVE-MGP always evaluates PATH $r(r+1)/2$ times. But we are interested in instances of the multi-goal planning problem where PATH is considerably more expensive to evaluate than TOUR. In robotics applications, this situation occurs frequently because $r$ is usually rather small, while $F$ is high-dimensional and geometrically complex. Hence, we wish to use PATH sparingly, even if this leads to evaluating TOUR (or just MIN-SPANNING-TREE) more often. Simultaneously, we would like the length of the generated multi-goal path to remain within a bounded factor of $\mathcal{L}_N$.

# 4 Greedy Planning Algorithm

To reduce the number of PATH evaluations, we create the goal graph $G = (V, X, c)$ with the cost $c(g_i, g_j)$ of every edge $\{g_i, g_j\}$ initialized to the length of the shortest path joining $g_i$ and $g_j$ in $C$. Since the shortest path in $C$ may not be collision-free, this length is a lower-bound approximation of $\ell(\text{PATH}(g_i, g_j))$. Usually, this lower bound is very easy to calculate, as for many measures $\ell$ the shortest path in $C$ between two configurations is the straight line segment joining them. Then, we successively adjust edge costs by evaluating PATH for edges picked in the current minimum spanning tree of $G$. Once the cost $c(g_i, g_j)$ of an edge $\{g_i, g_j\}$ has been updated to $\ell(\text{PATH}(g_i, g_j))$, this edge $\{g_i, g_j\}$ is said to be *exact*. This yields the greedy algorithm GREEDY-MGP given in Figure 4 and discussed below.

**Algorithm** GREEDY-MGP
1. Set $G$ to $(V, X, c)$
2. $V \leftarrow \{g_0, g_1, ..., g_r\}$
3. $X = \{\{g_i, g_j\} \,|\, i, j \in \{0, 1, ..., r\}\}$
4. For every $e = \{g_i, g_j\} \in E$ set $c(e)$ to the length of the shortest path in $C$ between $g_i$ and $j_j$
5. Repeat
   a. $T \leftarrow$ MIN-SPANNING-TREE($G$)
   b. $\kappa \leftarrow c(T)$
   c. Repeat while $c(T) \leq \alpha \times \kappa$
      i. If all edges in $T$ are exact then return multi-goal path determined by PREORDER-WALK($T$)
      ii. Pick a non-exact edge $e = \{g_i, g_j\}$ in $T$
      iii. Reset $c(e)$ to $\ell(\text{PATH}(g_i, g_j))$

Figure 4: Greedy multi-goal planning algorithm

Steps 1 through 4 initialize the goal graph $G$. Hence, just after performing Step 4, all edges are non-exact. Step 5 first computes the minimum spanning tree $T$ of $G$. Next, it evaluates PATH for non-exact edges of $T$, until either all edges are exact (in which case it exits with a solution), or the new cost of the tree is greater than $\alpha$ times its original cost. In the latter case, it loops by computing the new minimum spanning tree. (If PATH ever fails to find a path at Step 5.c.iii, then GREEDY-MGP returns *failure*.)

For a given multi-goal planning problem, let $\mathcal{L}_N$ and $\mathcal{L}_G$ denote the lengths of the solutions computed by NAIVE-MGP and GREEDY-MGP, respectively. Assume that, whenever these two algorithms evaluate PATH for the same two goals $g_i$ and $g_j$, each evaluation yields the same goal-to-goal path. Then, we have:

**Claim 1:** *The lengths $\mathcal{L}_N$ and $\mathcal{L}_G$ of the multi-goal paths computed by* NAIVE-MGP *and* GREEDY-MGP *verify*

$$\mathcal{L}_G \leq 2\alpha \times \mathcal{L}_N.$$

*Proof:* Let $G_{ex}$ denote the goal graph in which all edges are exact, hence weighted by the lengths of the paths computed by PATH. Let $T_{ex}$ be the minimum spanning tree of $G_{ex}$. Recall that the length $\mathcal{L}_N$ of the multi-goal path computed by NAIVE-MGP verifies $c(T_{ex}) \leq \mathcal{L}_N \leq 2c(T_{ex})$.

During the evaluation of GREEDY-MGP, the cost of every edge of $G$ is always less than or equal to the cost of the same edge in $G_{ex}$. Thus, at Step 5, we have $\kappa \leq c(T_{ex})$. Step 5.c computes goal-to-goal paths for non-exact edges in $T$, until either the cost of $T$ grows larger than $\alpha \times \kappa$, or all edges in $T$ are exact. In this second case, we have $c(T) \leq \alpha \times \kappa \leq \alpha \times c(T_{ex})$. Since the length $\mathcal{L}_G$ of the multi-goal path determined by PREORDER-WALK($T$) at Step 5.c.i is guaranteed to be within factor 2 of $c(T)$, we have $\mathcal{L}_G \leq 2\alpha \times c(T_{ex})$, hence $\mathcal{L}_G \leq 2\alpha \times \mathcal{L}_N$. ■

Note that the tour PREORDER-WALK($T$) computed at Step 5.c.i may contain edges that are still not exact. For those edges, it is necessary to evaluate PATH in order to obtain a multi-goal path. Then, the goal graph may still contain many edges that are non-exact and the triangular inequality may not be satisfied by every triplet of nodes. But all what matters for the tour computed at Step 5.c.i to have cost within factor 2 of the cost of $T$ is that the triangular inequality holds for the edges in $T$ and its preorder walk.

In the worst case, GREEDY-MGP evaluates PATH $r(r+1)/2$ times. However, the experimental results of Section 6 show that in practice PATH is usually evaluated much fewer times.

# 5 Implementation of PATH

For robot arms, no algorithm is available to efficiently compute goal-to-goal paths that are within a guaranteed factor of shortest paths. Instead, we use a heuristic two-phase approach: first, compute a collision-free path; next, optimize this path. This approach was introduced in [1].

Our implementation of PATH uses the single-query PRM planner described in [13] to generate an initial collision-free path between two given goals $g_i$ and $g_j$. This planner works by incrementally growing two trees of sampled configurations, called milestones, that are rooted at $g_i$ and $g_j$, respectively. At every iteration, it picks a milestone $m$ at random in one of the two trees and samples configurations in a neighborhood of $m$ until one of them, $m'$, tests collision-free. It installs $m'$ as a child of $m$ in its tree and creates a connection between $m'$ and the closest milestone in the other tree, thus establishing a path of milestones between $g_i$ and $g_j$. The planner then checks that the connection between every two successive milestones on this path is collision-free (each connection is a straight line segment in configuration space). If all connections are collision-free, the planner returns the path, otherwise it keeps growing the trees of milestones. The planner exits with failure if it has not found a path after generating a given maximum number of milestones. The planner is probabilistically complete, with fast convergence rate [8, 13]. Failure to find a path, while one exists, has not been an issue in our experiments.

The milestone trees grown by the planner are not biased in any particular direction. So, rather than discarding the two trees produced by a run of the planner, we store them. When PATH is invoked again, if any of the two goals in the new pair of goals has already been considered before, then the corresponding tree is reused by the planner, thus saving considerable amount of work. As more evaluations of PATH are performed, each evaluation, on average, takes less time.

Several optimizers can be used to improve a path generated by the PRM planner. For instance, the variational optimization technique used in [1] iteratively deforms the path to minimizes the time needed to execute the path, given the robot's dynamic model and torque limits in the joints. Our implementation of PATH uses a simpler (and faster) optimizer. A path generated by the PRM planner is a polygonal line in configuration space. We use the $L_\infty$ metric weighted by the inverse of the maximum velocity of each joint to measure the length of each segment. Thus, the length of a path is the time to execute this path assuming that, along each segment, all joints move at constant velocity, with the joint needing the most time moving at maximum velocity. In practice, this length is only a lower-bound approximation of the actual time needed to execute the path. With this weighted $L_\infty$ metric, the shortest path in configuration space between any two configurations is the straight segment. The optimizer repeatedly replaces sub-paths by straight segments whenever these segments are collision-free. Though the outcome is at best locally optimal, it is usually quite satisfactory.

# 6 Experimental Results

## 6.1 Comparison with NAIVE-MGP

We first compare GREEDY-MGP and NAIVE-MGP on two examples shown in Figure 5, with 10 and 30 goals, respectively.

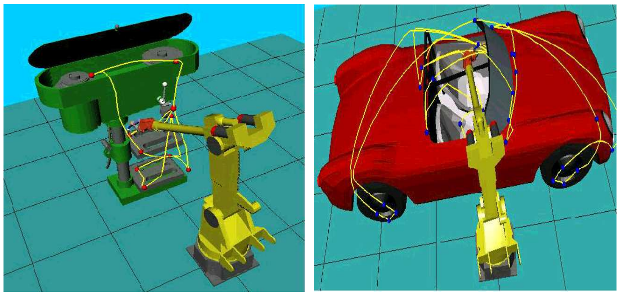

Figure 5: Examples 1 and 2 (10 and 30 goal configurations)

To facilitate the comparison, we proceed as follows. After every run of GREEDY-MGP we run NAIVE-MGP with the same seed of the random-number generator. In addition, during the run of GREEDY-MGP we note the sequence of pairs of goals on which PATH is invoked. Then NAIVE-MGP first computes goal-to-goal paths in this same order, before computing paths between all remaining pairs of goal (in any order). In this way, over the two runs, we are guaranteed that the same query PATH$(q_i, q_j)$ made by GREEDY-MGP and NAIVE-MGP yields the same goal-to-goal path.

| | NAIVE-MGP | | GREEDY-MGP | |
|---|---|---|---|---|
| | Ex. 1 | Ex. 2 | Ex. 1 | Ex.2 |
| Total-time | 47.8 | 325.13 | 15.2 | 52.67 |
| Total-length | 5.06 | 9.1 | 5.21 | 10.24 |
| #PATH | 55 | 465 | 19 | 52 |
| #TOUR/MST | 1 | 1 | 8 | 36 |
| Time-PATH | 48.2 | 317.7 | 10.2 | 15.3 |
| Time-TOUR/MST | 0.35 | 6.52 | 2.53 | 32.1 |

Table 1: Comparative performance results obtained with NAIVE-MGP and GREEDY-MGP ($\alpha = 1$)

Table 1 gives performance data obtained with NAIVE-MGP and GREEDY-MGP, when the parameter $\alpha$ is set to 1. The data are averages over 50 runs of each planner, with different seeds of the random-number generator (with every seed used to run both planners as indicated above). The columns correspond to the three examples. The rows indicate the total running times of a planner (in seconds), the length of the generated solution, the number of evaluations of PATH, the number of evaluations of MIN-SPANNING-TREE or TOUR, and the times spent in those evaluations.

### 6.2 Influence of parameter $\alpha$

Here, on each of the two previous examples, we measure the performance of GREEDY-MGP for several values of the parameter $\alpha$. Each of the two plots in Figure 6 corresponds to one example, and contains three curves that respectively show the total running time of GREEDY-MGP, the length of the computed multi-goal path, and the number of evaluations of PATH when $\alpha$ grows from 1 to 3. Again, each value is an average over 50 runs.

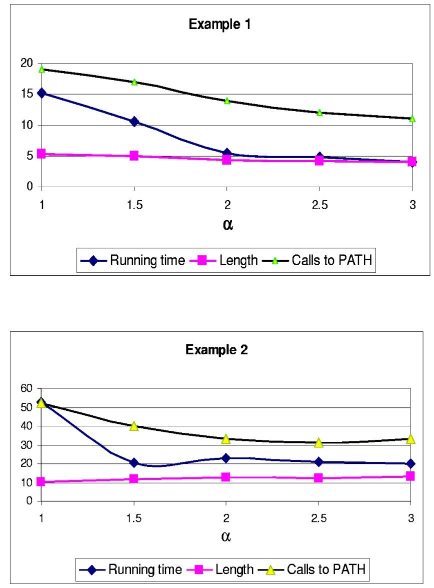


Figure 6: Running time of GREEDY-MGP, length of solution, and number of calls to PATH when $\alpha$ grows from 1 to 3

## 7 Conclusion

This paper describes a new multi-goal planning algorithm, GREEDY-MGP, that operates under the assumption that the cost of finding a good path between any two goals dominates that of finding a good tour in a graph with edges of given weights. Our algorithm uses the minimum spanning tree of the goal graph to decide which goal-to-goal paths to compute. The solution it returns is within $2\alpha$ (where $\alpha \geq 1$) of the cost of the solution returned by the naive algorithm that first computes all goal-to-goal paths. Experiments show that in general our algorithm is much faster than the naive algorithm or the algorithm previously proposed in [14]. Its running time also drops sharply when $\alpha$ is set to values slightly greater than 1. The approximation factor $2\alpha$ holds for the classic TSP algorithm that "doubles" the minimum spanning tree of the goal graph. Other TSP algorithms [10] might yield a more efficient GREEDY-MGP in practice, but we have not experimented with any them. Most would not guarantee a $2\alpha$ approximation factor.

There are several directions in which this work could be extended in the future. In some applications, a partial ordering is imposed on the input goals. How to efficiently incorporate this ordering into the multi-goal planner? In some cases, it is more convenient to specify the goals as configurations (positions and orientations) of the robot's end-effector. Since the inverse kinematics or a robot arm often has several solutions, each goal configuration of the end-effector may map into several goal configurations of the robot, but only one of these would have to be visited by the robot. Choosing among alternative robot goals would add to the complexity of the planner. Finally, it would be interesting to investigate the multi-robot case, where several robots sharing portions of their workspaces must reach goals without colliding with each other.


## Acknowledgments

This research was partially funded by a gift from General Motors Research and ABB. This paper has benefited from discussions with Prof. Fritz Prinz, Dr. Fabian Schwarzer and Yu-Chi Chang.


# References


[1] J.E. Bobrow. Optimal Robot Path Planning Using the Minimum-Time Criterion. *IEEE J. Robotics and Automation*, 4(4):443-450, August 1988.

[2] M. Bonert, L.H. Shu, and B. Benhabib. Motion Planning for Multi-Robot Assembly Systems. *Proc. ASME Design Engineering Technical Conferences*, Las Vegas, NV, Sept. 1999.

[3] J.F. Canny and J. Reif. New Lower Bound Techniques for Robot Motion Planning Problems. *Proc. IEEE Conf. on Foundations of Computer Science*, pp. 39-48, 1987.

[4] B. Cao, G.I.Dodds, and G.W. Irwin. A Practical Approach to Near Time-Optimal Inspection-Task-Sequence Planning for Two Cooperative Industrial Robot Arms. *Int. J. of Robotics Research*, 17(8):858–867, 1998.

[5] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*, The MIT Press, Cambridge, MA, 1990.

[6] T. Danner and L.E. Kavraki. Randomized Planning for Short Inspection Paths. *Proc. IEEE Int. Conf. on Robotics and Automation*, San Francisco, CA, April 2000.

[7] H.H. Gonzalez-Baños and J.C. Latombe. A Randomized Art-Gallery Algorithm for Sensor Placement. *Proc. ACM Symp. on Computational Geometry*, pp. 232-240, 2000.

[8] D. Hsu. *Randomized Single-Query Motion Planning in Expansive Space.* Ph.D. Thesis, Computer Science Dept., Stanford University, Stanford, CA, May 2000.

[9] L.E. Kavraki, P. Svetska, J.C. Latombe, and M. Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Tr. on Robotics and Automation*, 12(4):566-580, 1996.

[10] D.S. Johnson and L.A. McGeoch. The Traveling Salesman Problem: A Case Study in Local Optimization. In *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra (eds.), John Wiley and Sons, London, UK, pp. 215–310, 1997.

[11] J.S.B. Mitchell. Shortest Paths and Networks. In *Discrete and Computational Geometry*, J.E. Goodman and J. O'Rourke (eds.), CRC Press, Boca Raton, FL, pp. 445–466, 1997.

[12] D.J. Rosenkrantz, R.E. Stearns, and P.M. Lewis II. An Analysis of Several Heuristics for the Traveling Salesman Problem. *SIAM J. of Computing*, 6(3):563–581, 1977.

[13] G. Sánchez and J.C. Latombe. On Delaying Collision Checking in PRM Planning – Application to Multi-Robot Coordination. *Int. J. of Robotics Research*, Vol.21, No.1, pp. 5-26, 2002.

[14] S.N. Spitz and A.A.G. Requicha. Multiple-Goals Path Planning for Coordinate Measuring Machines. *Proc. IEEE Int. Conf. Robotics and Automation*, San Francisco, CA, 2000.

[15] C. Wurll, D. Henrich, and H. Wörn. Multi-Goal Path Planning for Industrial Robots. *Proc. Int. Conf. on Robotics and Applications (RA'99)*, Santa Barbara, CA, 1999.