Coverage of a Planar Point Set with Multiple Robots subject to Geometric Constraints

Nilanjan Chakraborty, *Student Member, IEEE,* Srinivas Akella, *Member, IEEE,* and John T. Wen, *Fellow, IEEE*

***Abstract*—This paper focuses on the assignment of discrete points among K robots and determination of the order in which the points should be processed by the robots, in the presence of geometric and kinematic constraints between the robots. This is a path planning problem that arises as a subproblem in the decoupled approach to solving the motion planning problem of covering a point set by a multiple robot system in minimum time. More concretely, our work is motivated by an industrial microelectronics manufacturing system with two robots, with square footprints, that are constrained to translate along a line while satisfying proximity and collision avoidance constraints. The N points lie on a planar base plate that can translate along the plane normal to the direction of motion of the robots. The geometric constraints on the motions of the two robots lead to constraints on points that can be processed simultaneously.**

We use a two step approach to solve the path planning problem: (1) *Splitting Problem*: Assign the points to the K robots, subject to geometric constraints, to maximize the parallel processing of the points. (2) *Ordering Problem*: Find an order of processing the split points by formulating and solving a multi-dimensional Traveling Salesman Problem (TSP) in the K-tuple space with an appropriately defined metric to minimize the total travel cost. We show that for $K = 2$, the splitting problem can be converted to a maximum cardinality matching problem on a graph and solved optimally in polynomial time. The matching algorithm takes $O(N^3)$ time in general and is too slow for large datasets. Therefore, we also provide a greedy algorithm for the splitting problem that takes $O(N \log N)$ time. We provide computational results comparing the two approaches and show that the greedy algorithm is very close to the optimal solution for large datasets. We also provide computational results for the ordering problem and present local search based heuristics to improve the TSP tour. Further, we give computational results showing the overall performance gain obtained (over a single robot system) by using our algorithm. Finally, we extend our approach to a K-robot system and give computational results for $K = 4$.

Note to Practitioners—**This paper presents techniques to plan the motions of multiple robots to visit and process a given set of points, subject to geometric constraints on the robot motions. This point set coverage task is motivated by a laser drilling application for electronics manufacturing. The goal is to minimize the time taken to visit the points by parallelizing the robot operations while avoiding robot collisions. Similar tasks arise in many applications including drilling, electronics manufacturing, circuit testing, spot welding, and sensor network data collection. We model the assignment of points in the plane to robots as a matching problem and the point traversal order generation as a Traveling Salesmen Problem. We present effective algorithms to plan the motions of the robots for large data sets (involving hundreds of thousands of points), and demonstrate the feasibility of our approach for 2 and 4 robots.**

***Index Terms*—Multiple-robot systems, point set coverage, matching, K-TSP.**

Nilanjan Chakraborty and Srinivas Akella are with the Department of Computer Science at Rensselaer Polytechnic Institute (Email: chakrn2@cs.rpi.edu; s.akella@ieee.org). John Wen is with the Department of Electrical, Computer, and Systems Engineering at Rensselaer Polytechnic Institute (Email: wenj@rpi.edu). All authors are also with the Center for Automation Technologies and Systems (CATS) at RPI. This work is supported in part by CATS under a block grant from the New York State Office of Science, Technology, and Academic Research (NYSTAR). John Wen is supported in part by the Outstanding Overseas Chinese Scholars Fund of Chinese Academy of Sciences (No. 2005-1-11). Nilanjan Chakraborty and Srinivas Akella were supported in part by NSF under CAREER Award No. IIS-0093233.

The corresponding author is Srinivas Akella, Department of Computer Science, Rensselaer Polytechnic Institute, 110 Eighth Street, Troy, New York 12180, USA. Tel: (518) 421-0800, Email: s.akella@ieee.org.

I. Introduction

Robotic point set coverage tasks occur in a variety of application domains like electronic manufacturing (laser drilling [4], inspection [3], circuit board testing [22], [11]), automobile spot welding [15], and data collection in sensor networks [18]. The goal of using multiple robots in point set coverage tasks is to reduce the overall task completion time by parallelizing the operations at the points. The path planning problem in such multi-robot point set coverage tasks can be stated as follows: *Given a point set $S = \{\mathbf{p}_i\}$, $i = 1, \ldots, N$, and K robots, find an assignment of the points to individual robots and determine the order in which the robots must visit the points so that the overall task completion time is minimized*. In this paper we look at such path planning problems for multiple robot point set coverage where the robots have to (a) spend some time at each point to complete a task (we call this time the *processing time* of each point) and (b) satisfy given geometric constraints (like collision avoidance) while covering the point set. Our work is motivated by a system (see Figure 1) used by a microelectronics manufacturer for laser drilling. Here we need to process (drill) a set of points with identical processing times by a system of $K(= 2)$ robots. The architecture of the machine imposes the following geometric constraints: (a) at any instant of time, each robot can process exactly one point within a square region in the plane (called *processing footprint*) although there may be several points within the region, (b) the robots are constrained to move along a line while avoiding collisions, and (c) the points lie on a base plate that can translate along the y-axis.

In the absence of the geometric constraints and assuming the processing times to be zero, the path planning problem for multi-robot point set coverage tasks can be treated as a K-Traveling Salesman Problem (K-TSP). However the path planning problem with inter-robot geometric constraints has not received much attention in the literature. In this paper, we provide solution algorithms to the path planning problem for

point set coverage with multiple robots, where the robots are subject to geometric constraints, and the processing time of each point is identical. We divide the problem into that of finding the assignment of points to each robot, and the order in which they will be processed while satisfying the geometric constraints as follows:

Splitting Problem: *Let P be a set of subsets of S of size less than or equal to K such that each point in S can occur in exactly one element of P*[1]*. The splitting problem is to find a set P of minimum cardinality that respects the geometric constraints such that each point in S occurs in exactly one element of P. Intuitively, such a set allows the maximum possible parallelization of the processing operation.*

Ordering Problem: *Given a set of K-tuples, P, find a processing order of the points by the robots such that the* cost *of visiting all the points is minimized.*

The splitting problem can be reduced to a clique partitioning problem [9] on a graph for arbitrary K and the ordering problem can be reduced to a traveling salesman problem (TSP) [1]. Both these problems are NP-hard. However, for two robots (i.e., $K = 2$) we show that the splitting problem is equivalent to a maximum cardinality matching (MCM) problem on a suitably constructed graph and can thus be solved optimally in polynomial time. The maximum cardinality matching algorithm takes $O(N^3)$ time in general ([12], [14]) and is not suitable for large data sets (in our application, approximately 10^5 points). Therefore, we provide a greedy algorithm that exploits the geometric nature of the constraints and takes $O(N \log N)$ running time. We provide results comparing the greedy algorithm with the matching algorithm for small datasets. Our computational experiments show that for typical industrial datasets the greedy algorithms give solutions that are very close to the optimal solution.

We model the ordering problem as a multi-dimensional TSP ([11], [22]) in the set of point pairs (pair space) obtained from the splitting problem. The solution of this TSP gives tours for the individual robots. We identify necessary conditions in the tours of the individual robots that improves the tour in the pair space. We provide results showing an improvement of 5% to 8% in the TSP tour using the tour improvement heuristic. To include points in the tour that were not paired in the splitting stage we give a cheapest insertion heuristic. We also give computational results showing the improvement of performance obtained by using a two robot system and the path planning algorithms described above over a single robot system. Finally, we extend our algorithms for the two robot system to a K robot system and provide some computational results for a four robot system.

This paper is organized as follows: In Section II, we briefly summarize the relevant literature. In Section III we formulate the path planning problem for the K-robot system and outline division of the problem into two subproblems. In Section IV, we provide solution algorithms and results for the two robot point splitting problem. In Section V, we formulate the ordering problem as a TSP in the pair space and provide local search based heuristics to improve the tour. In Section VI, we extend the algorithms for the two robot system to a K-robot system. Finally, in Section VII, we present our conclusions and outline future work. A preliminary version of this work appeared in [4].

[1]We will refer to each element of P as a K-tuple with the understanding that if there are less than K points present in an element we add virtual points to make its cardinality equal to K.

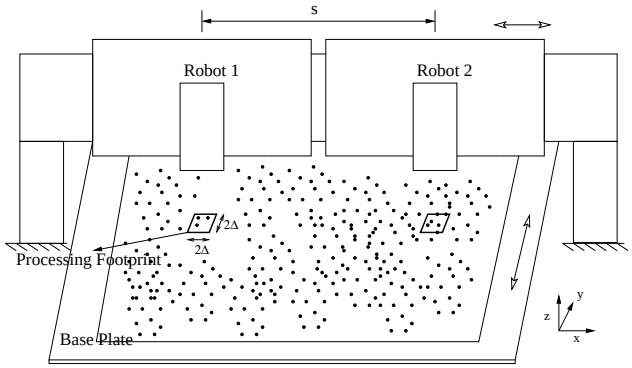

Fig. 1. Schematic sketch of a 2-robot system used to process points in the plane. The heads can translate along the x-axis and the base plate translates along the y-axis. The square region of length 2Δ is the processing footprint for each robot.

II. Related Literature

The general motion planning problem for the minimum-time multiple robot point set coverage problem can be stated as follows:

Input: *A set of points $S = \{\mathbf{p}_i\}$ with processing times τ_i, $i = 1, 2, \ldots N$, that must be processed by K robots where each robot has a limited footprint and the robots must satisfy given geometric, kinematic, and dynamic constraints.*

Output: *A trajectory for each robot satisfying the constraints such that the total time (process time plus travel time) required to cover the point set is minimized.*

This is a hybrid discrete-continuous optimization problem because we have to simultaneously optimize over (a) the feasible discrete choices involved in the assignment of points to the robots and the order in which the points are visited by the robots, (b) the feasible continuous choices involved in specifying the position and velocity of the robots as a function of time. The problem is hard to solve even for $K = 1$, even without the geometric constraints.

There are two distinct approaches to solving hybrid discrete-continuous optimization problems like the one above: (1) Form a mixed integer nonlinear optimal control problem [20] or (2) Use a two stage approach: (a) Solve the discrete optimization problem of finding the path and (b) Solve the continuous optimization problem of converting the path into a trajectory. The first approach is very general although the resultant problem is very hard to solve in practice. von Stryk and Glocker [20] used this approach to find the trajectories of two cooperating robots (cars), with given kinematic motion model, visiting a set of points. They used a two level iterative scheme to find the optimal trajectories. The outer level iteration used a branch and bound framework to search the space of discrete variables (in this case, the variables corresponding to assignment and order of the points). The inner level iteration solved a nonlinear optimal control problem over the space of continuous variables

(in this case, the position and velocity of the robots). This approach can require the solution of an exponential (in the number of points to be visited) number of inner level nonlinear optimal control problems, each of which is nontrivial to solve. Hence, this approach is limited to a small number of points.

The literature for the second approach usually focuses on either the discrete optimization problem of covering a point set or the continuous optimization problem of trajectory generation. The problem of covering a point set by a single robot with collision avoidance constraints has been studied for industrial robots [15], [21], [17], [2]. Saha $et\ al.$ [15], and Wurll and Henrich [21] address motion planning of a fixed base manipulator to process a set of points avoiding static obstacles in the workspace. The points are assumed to be partitioned into groups and the motion is assumed to be point-to-point. In [17], Spitz and Requicha consider the point set processing problem for a coordinate measuring machine. Since there is only one robot, the processing time is constant and the main focus of these papers is to find a minimum cost collision free path covering all the points. The collision avoidance problem is nontrivial in these cases and all of the above papers use a discrete search of the configuration space ([15] and [17] use different versions of probabilistic roadmaps whereas [21] uses A^* search) for computing a collision free path. On the other hand, we have multiple robots and algebraic equations that give collision avoidance constraints. Thus we focus on assigning the points to the robots (to reduce processing time) as well as obtaining an order of processing them (to reduce traveling time) while avoiding collisions among the robots.

Dubowsky and Blubaugh (see [6], Section IV) considered the problem of multiple manipulators processing a set of points. However, they assumed that the manipulators will never be in collision with each other and formulated the problem as a K-TSP. Their solution approach was to find a tour for one manipulator and then divide it into K tours for K manipulators such that the maximum of the K tour costs is minimized. Here, we need to satisfy collision avoidance constraints, and such an approach is not suitable.

III. Problem Formulation

The motion planning problem for minimum time multiple robot point set coverage can be formulated as a mixed integer optimal control problem (see Appendix A). The solution of this formulation, when it can be solved, gives the trajectories of the K robots such that the overall time taken in covering the point set is minimized. However, as observed in [20], it is difficult to solve problems of this type directly and this is especially true for the large datasets that we have to deal with. Therefore, we follow the usual approach in the robotic motion planning literature and divide the motion planning problem into a path planning problem and trajectory optimization problem. In this paper we are concerned with the formulation and solution of only the path planning problem. There are two main questions that arise in the formulation of the path planning problem:

1) Do we need to incorporate the inter-robot geometric constraints in the path planning problem?
2) What is a measure of the travel cost in the presence of the inter-robot geometric constraints?

Before we proceed to answer the above two questions we first look at the minimum cost path planning problem for K robots, without geometric constraints, where the robots have some processing time at each point. If the processing times are all zero then the problem is equivalent to a K-TSP problem defined below:

Given a weighted complete graph, $G = (V, E)$, where the set of vertices V consists of the set of all the points and the weight on an edge is the travel cost between the two points, find K subtours on this graph such that the cost of the most expensive subtour (i.e., sum of weights of the edges in the subtour) among the K subtours is minimized.

If the processing times are non-zero the problem can still be set up as a K-TSP with suitable weights. Let $S = \{\mathbf{p}_i\}, i = 1, \ldots, N$, be the point set that is to be processed in minimum time by K robots where each robot has to spend time τ_i at point $\mathbf{p}_i$. Let d_{ij} be the travel cost on the edge (i, j), τ_i, τ_j, be the processing times of points $\mathbf{p}_i$, $\mathbf{p}_j$ respectively and $w > 0$ be an arbitrary positive number[2]. Define the new cost on the edges as

$$\bar{d}_{ij} = d_{ij} + \frac{w(t_i + t_j)}{2} \tag{1}$$

The solution of a K-TSP with the edge weights defined above gives the tours for each robot such that the maximum sum of processing cost and travel cost is minimized. Moreover, if d_{ij} satisfies the triangle inequality then $\bar{d}_{ij}$ also satisfies the triangle inequality. Consider three vertices i, j, k. From Equation 1 we have:

$$\begin{aligned} \bar{d}_{ij} + \bar{d}_{jk} &= d_{ij} + \frac{w(t_i + t_j)}{2} + d_{jk} + \frac{w(t_j + t_k)}{2} \\ &= d_{ij} + d_{jk} + wt_j + \frac{w(t_i + t_k)}{2} \\ &\geq d_{ik} + \frac{w(t_i + t_k)}{2} \geq \bar{d}_{ik} \end{aligned} \tag{2}$$

Thus the new cost metric defined by Equation 1 satisfies the triangle inequality. Therefore, we can use the algorithm given in [8] to obtain a constant factor approximation algorithm for the problem. The constant factor is $5/2$ if we use Christofides algorithm to solve the 1-TSP problem.

In the presence of inter-robot geometric constraints, the solution obtained above by ignoring the constraints may be arbitrarily bad, i.e., the K-TSP solution may result in no parallelization of the operations. We illustrate this with a simple example. Figure 2 shows a simple input point distribution where no two points lie simultaneously in the processing zone of a single robot. Assume the geometric constraints on the robot is the same as that in Figure 1. The bold line shows the optimal paths of the two robots obtained ignoring the geometric constraints. However, because of the constraints, no two points assigned to the two robots in this solution can be processed simultaneously and the time taken is the same as that would be achieved by a single robot. Thus, solving a

[2] w is a problem domain dependent scale factor that accounts for different units that the travel cost and processing cost may have, e.g., travel cost may have units of length whereas processing cost may have units of time.

standard K-TSP ignoring the geometric constraints can give an arbitrarily bad solution. Thus the answer to our first question is that we indeed do need to take the inter-robot geometric constraints into consideration at the path planning stage. To answer the second question, we first note that in the presence of geometric constraints the maximum cost subtour among the K subtours is not the right measure of the overall cost. This is because the inter-robot constraints imply that the robots may not be able to simultaneously traverse parts of their own tours. The total travel cost incurred for task completion in this case is the sum of the simultaneous travel costs of the robot and the individual travel cost of the robots ($i.e.$, costs of parts of the tour where they cannot move simultaneously).

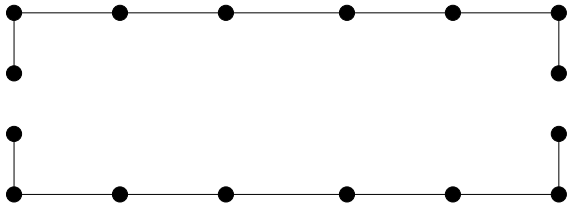

Fig. 2. An example input distribution of points where the bold lines show an optimal 2-TSP tour on this set obtained ignoring the geometric constraints. Clearly, the two robots for the system in Figure 1 cannot process any pair of points simultaneously while satisfying the geometric constraints.

Although we need to take care of the inter-robot collision avoidance constraints, it is not clear how to do that at the path planning stage because these constraints should be satisfied at all times whereas there is no time information in the path planning problem. Therefore, we pose the path planning problem in the space of K-tuples of points and define the *feasible path* for this problem in the K-tuple space as follows: *A feasible path is defined as an ordered set of tuples of points of size less than or equal to K such that the robots satisfy the inter-robot geometric constraints if they are present at points in the same tuple*. The path in the K-tuple space induces a path in the Euclidean space for each robot. This definition ensures that when the robots move along their respective paths, they would satisfy the geometric constraints if they are at the points belonging to the same tuple at the same time. Thus, once we have an ordered set of tuples, we can ensure that the geometric constraints are satisfied when moving along the path by writing the constraints as simple velocity constraints of the robots in the point-to-point motion trajectory optimization problem. To solve the above path planning problem, we take the following two step approach:

1) Divide the set of points into K-tuples of points that satisfy the geometric constraints and assign the points in each tuple to a robot.
2) Find the order in which the tuples should be processed by the robots (this produces a tour for each robot on the set of points assigned to it).

We can also view the two step approach as follows: we first find a collection of points that can be processed simultaneously, thus obtaining an optimal solution for the processing time and then we find a travel path for the robots that minimizes the travel cost without changing the processing cost. Therefore, in cases where the processing time dominates the travel time this approach will give a good solution to the overall problem.

IV. Splitting Problem

In this section we look at the splitting problem for the two robot system shown in Figure 1. The splitting problem consists of assigning the set of points to the robots so as to maximize the number of points that can be processed together. We call a pair of points a *compatible pair (of points)* if they can be processed together while respecting the geometric constraints. Any two compatible pairs are called a *disjoint compatible pair* if the points belonging to the two pairs are distinct. Thus, the splitting problem is equivalent to minimizing the total number of disjoint compatible pairs and singletons while assigning them to the robots. A solution to this problem ensures the maximum parallelization of the processing operation. The formal statement for this problem is given below:

Problem Statement: Let $S = \{\mathbf{p}_i\} = \{(x_i, y_i)\}_{i=1}^{N}$, be a set of points in $\mathbb{R}^2$. Let P be a set of *ordered* subsets of S of size less than or equal to 2 that partitions S, i.e., $P = \{(\mathbf{p}_i, \mathbf{p}_j)\} \bigcup \{(\mathbf{p}_k, *)\} \bigcup \{(*, \mathbf{p}_l)\}$, $i, j, k, l \in \{1, 2, \ldots, N\}, i \neq j \neq k \neq l$, where $*$ denotes a *virtual point* and any pair $(\mathbf{p}_i, \mathbf{p}_j) \in P$ respects the following constraints

$$\begin{aligned} |x_i - x_j| &\geq s_{\min} - 2\Delta \\ |y_i - y_j| &\leq 2\Delta \end{aligned} \tag{3}$$

where $s_{\min}$ is the minimum distance between the two robots. Find such a P of minimum cardinality.

In the above statement the ordered pair $(\mathbf{p}_i, \mathbf{p}_j)$ denotes that $\mathbf{p}_i$ is assigned to robot 1 and $\mathbf{p}_j$ is assigned to robot 2. Moreover $(\mathbf{p}_k, *)$ denotes that $\mathbf{p}_k$ is a singleton assigned to robot 1, while $(*, \mathbf{p}_l)$ denotes that $\mathbf{p}_l$ is a singleton assigned to robot 2. The constraint between the x-coordinates of the points in Equation 3 ensures collision avoidance between the robots. The constraint on the y-coordinates indicate that the robots are constrained to move along the x-axis but have a square footprint for processing.

A. Optimal Algorithm

The problem above can be solved optimally by converting it to a maximum cardinality matching problem on a graph.

Definition Let $G = (V, E)$ be a graph where V is the set of vertices and E is the set of edges. A set $M \subseteq E$ is called a *matching* if no two edges in M have a vertex in common. M is called a *maximal matching* if there is no matching M' such that $M \subset M'$. M is called a *maximum cardinality matching* (MCM) if it is a maximal matching of maximum cardinality.

Definition A vertex in V is called a *matched vertex* if there is one edge incident upon it in M, otherwise it is called an *unmatched* or *exposed vertex*.

From the given set of N input points, S, we construct a graph $G = (V, E)$, where V is the set of all points and E is the set of all edges with an edge existing between two points iff they form a compatible pair, i.e., they satisfy Equation 3.

We call this graph the *compatibility graph* of the point set. A maximum cardinality matching on the compatibility graph gives the maximum number of disjoint compatible pairs, i.e., the maximum number of points that can be processed together. The unmatched vertices form the singletons that are to be processed individually. The set P consisting of the matched pairs and unmatched vertices will be of minimum cardinality since the number of singletons are minimum. After we obtain the matching solution, we can use the geometry of our problem to assign the points to the robots. In our problem, robot 1 is always constrained to be on the left of robot 2. Therefore, we order the pairs in the matching so that the point with lower x-coordinate is on the left and thus gets assigned to robot 1. For the singletons, we assign points on the left of the median of the point distribution to robot 1 and points on the right to robot 2.

The MCM problem on a graph is a well studied combinatorial optimization problem and can be solved in $O(N^3)$ time (Edmonds [7]). However, slightly faster algorithms do exist ($e.g.$, Micali and Vazirani's $O(\sqrt{|V|}|E|)$ algorithm [13]). In our application, N can be of the order of 10^5 and the matching algorithm is not practical for such large values of N. Hence we provide a greedy algorithm that gives a suboptimal solution and runs in $O(N \log N)$ time. We note that although there are greedy algorithms in the matching literature that have linear running time in the number of edges (see [19], and references therein), such algorithms assume the input to be available in the form of a graph. In our problem, the input is a set of points, S, and the parameters Δ and $s_{\min}$ specifying the geometric constraints. Hence, we need to construct the input graph from this information and this may take $O(N^2)$ time in the worst case.

B. Greedy Algorithm

Given the set of input points, S, and the parameters, Δ and $s_{\min}$, we use the geometric structure of our constraints and the distribution of the input points to design a greedy algorithm. We first divide the points along the y-axis into bands of width 2Δ and then divide the points within each band into two almost equal halves using the median of the x-coordinates of the points in the band. Then starting from the left most point of the left half, we pair each point in the left half with a compatible point in the right half with minimum x-coordinate, breaking ties by choosing points with minimum y-coordinate. This is the best possible local choice for a point in the sense that this choice leaves the maximum number of compatible points on the right half for the remaining points on the left half. Algorithm 1 gives a detailed description of our greedy heuristic. The main computational cost in the algorithm is the sorting of the points according to their y coordinates at line 3 of Algorithm 1. Hence, the algorithm has a theoretical worst case running time of $O(N \log N)$.

We have divided the points into horizontal bands and then divided each horizontal band into two halves. These two choices imply that for every point we are restricting our choice of the points it can be paired with (or we are restricting the set of neighbors in the compatibility graph). Theoretically, this may seem to be troublesome and one can devise inputs where this scheme will perform badly. However, for the practical inputs this scheme works very well as shown in Table I. Moreover, one can also deal with this by repeating the algorithm with different starting y-coordinates for the bands between $y_{\min}$ and $y_{\min} + 2\Delta$ and taking the best result among them. Another alternative is that instead of dividing the point sets, we consider for each point all the possible unassigned points it can be paired with and use the same criterion as in Algorithm 1 for choosing a partner for this point. This algorithm is a special case of the greedy algorithm for matching where we pick one edge at random from the graph, remove all edges corresponding to the picked vertices and then pick another edge from the graph. The ratio of the number of edges in the optimal matching to the number of edges returned by this algorithm is upper bounded by 2. Thus the ratio of the total processing time of the points paired with this algorithm to the optimal processing time is $3/2$ assuming a constant processing

Algorithm 1 Greedy algorithm for 2 robots

1: *Input: Vector of x and y coordinates of points,* **x**, **y***; Parameters* $s_{\min}$, Δ. *[***x**, **y***] denotes the concatenated vectors* **x** *and* **y**.
2: *Output: Set P of subsets of S of cardinality* ≤ 2 *that partitions S.*
3: [**x**, **y**] = ysort(**x**, **y**); // *Sort according to y-coordinates*
4: $y_{\min}$ = minimum(**y**); $y_{\max}$ = maximum(**y**);
5: $numbands = \lceil \frac{y_{\max} - y_{\min}}{2\Delta} \rceil$ // *Number of bands*
6: **for** $i = 0$ to $numbands - 1$ **do**
7: [**u**, **v**] $\leftarrow$ Points with $y-$coordinates in the range ($y_{\min} + 2i\Delta, y_{\min} + 2(i+1)\Delta$)
8: [**u**, **v**] = xsort(**u**, **v**)
9: u_{div} = xmedian(**u**) // *Median of* $x-$*coordinates*
10: $u_{\min}$ = minimum(**u**); $u_{\max}$ = maximum(**u**);
11: **if** $|u_{\text{div}} - u_{\min}| < s_{\min}$ **then**
12: $u_{\text{div}} \leftarrow u_{\min} + s_{\min}$
13: **end if**
14: [$\mathbf{u}_l$, $\mathbf{v}_l$] $\leftarrow$ Points with $x-$coordinates in the range ($u_{\min}$, u_{div})
15: [$\mathbf{u}_r$, $\mathbf{v}_r$] $\leftarrow$ Points with $x-$coordinates in the range (u_{div}, $u_{\max}$)
16: **for** $j = 1$ to length($\mathbf{u}_l$) **do**
17: $k \leftarrow$ Index of point on right hand side that has minimum $x-$coordinate among all points that respect the constraints. If there is more than one such point we take the one with the minimum $y-$coordinate.
18: **if** such a k exists **then**
19: P $\leftarrow$ P $\bigcup$ $\{((u_l[j], v_l[j]), (u_r[k], v_r[k]))\}$
20: **else**
21: P $\leftarrow$ P $\bigcup$ $\{((u_l[j], v_l[j]), *)\}$
22: **end if**
23: **end for**
24: **end for**
25: $L \leftarrow$ Set of indices of unassigned points on right side
26: P $\leftarrow$ P $\bigcup$ $\{(*, (u_r[k], v_r[k]))\}, \forall k \in L$
27: **return** P

TABLE I
PERFORMANCE COMPARISON OF SPLITTING BETWEEN GREEDY AND MATCHING ALGORITHM

	Greedy Algorithm			Matching Algorithm		
Number of points	Number of pairs	Number of singletons	Time (sec)	Number of pairs	Number of singletons	Time (sec)
1396	695	6	0.45	696	4	5
11109	3029	5051	0.97	4137	2835	94
27810	13840	130	5.17	13905	0	1528
135300	67649	2	107			
167536	83739	58	172			
181758	90866	26	217			
198570	99279	12	246			
211856	105845	166	288			

time for each point. However, in experiments on the practical datasets this algorithm performs worse than Algorithm 1 and takes more time. A working scheme that can be used is to run both the algorithms and take the best of the two. This ensures that we always have a splitting result that is practically good and also has a worst case theoretical bound.

C. Results

The results of using both the greedy algorithm and optimal (matching) algorithm for the splitting problem, along with the corresponding running times, are shown in Table I. The value of the parameters used for obtaining the results are $\Delta = 8$ mm, $s_{\min} = 96$ mm. We have used an implementation of Edmond's algorithm available in the Boost Graph Library [16] to solve the MCM problem. The datasets used represent typical datasets that are used in the industry. Table I shows that for smaller datasets (say less than 30000 points), although the matching algorithm performs better, the running time is much higher and hence it is not practical to use it for large datasets. In fact, the Boost Graph implementation fails for large datasets (resulting in the blanks in Table I for larger datasets). Moreover, our computational experiments (last five rows of Table I) show that the ratio of the number of singletons to the number of points is very small, hence for practical purposes the greedy algorithm performs quite well. Figures 3 and 4 show two example datasets and the assignment of paired points to the two robots. The units of length on the two axes are microns. In Figure 4, the spread of the dataset along the x-axis is approximately 120 mm. As the minimum distance to be maintained between the two robots $s_{\min} = 96$ mm, we cannot process most of the points in parallel and consequently there are a large number of singletons in the middle.

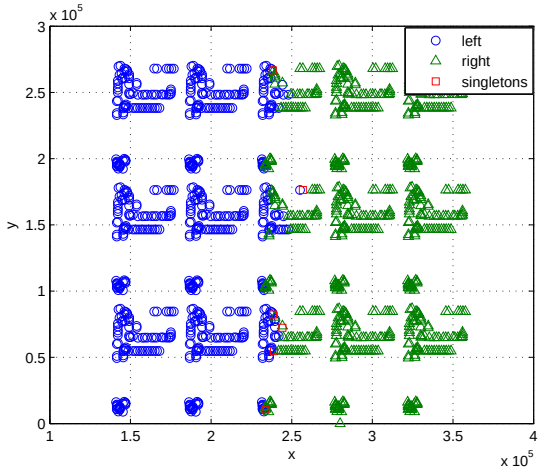

Fig. 3. Splitting and assignment of points by greedy algorithm for the dataset of 1396 points.

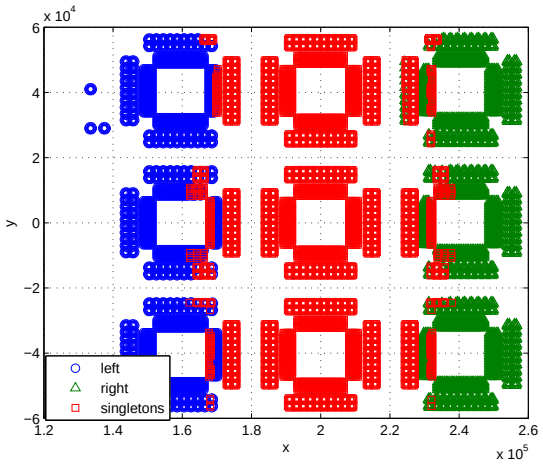

Fig. 4. Splitting and assignment of points by greedy algorithm for the dataset of 11109 points.

V. ORDERING PROBLEM

In this section we present algorithms to find an order of processing the points that minimizes the travel cost while ensuring that the compatible points are processed simultaneously. We formulate the problem as a multi-dimensional TSP (such TSP's also arise in the circuit testing literature [11], [22]). We use a three step approach to solve this problem: (1) Find a path through the compatible pairs by solving a TSP on the set of point pairs (pair space). The solution of this TSP in the pair space induces a tour for each robot in $\mathbb{R}^2$. Note that even if we find the optimal tour in the pair space the optimality is with respect to the given pairing of the points and it may be possible that another *feasible* pairing of the same cardinality gives a better tour. (2) Use a local search heuristic in the tour of each robot to find a better tour in the pair space while respecting the constraints. (3) Incorporate the singletons to be processed by each robot in this tour by using a *cheapest insertion heuristic*.

A. TSP in Pair Space (PTSP)

For formulating the TSP in the pair space or pair TSP (PTSP) we have to first define a metric in the pair space between two pairs that is meaningful to our problem. Since the relative motion of the robot and the points are constrained to be only along the x-axis and y-axis, a natural measure of distance, d_{ij}, between two points $[(x_{i1}, y_{i1}), (x_{i2}, y_{i2})]$ and $[(x_{j1}, y_{j1}), (x_{j2}, y_{j2})]$ in the pair space is given by:

$$\max(|x_{i1} - x_{j1}|, |x_{i2} - x_{j2}|, |y_{i1} - y_{j1}|, |y_{i2} - y_{j2}|) \quad (4)$$

This distance gives the cost incurred for robot 1 to reach (x_{j1}, y_{j1}) from (x_{i1}, y_{i1}) and robot 2 to reach (x_{j2}, y_{j2}) from (x_{i2}, y_{i2}) simultaneously. This distance measure is symmetric and satisfies the triangle inequality. The formal problem statement for PTSP can thus be written as:

Problem Statement: Given a set of pairs of points, $P = \{[(x_1, y_1), (x_2, y_2)]_i\}_{i=1}^{m}$, and a distance defined on the pairs by Equation 4, find a minimum cost tour on the weighted complete graph $G = (V, E)$, where $V = \{1, 2, \ldots, m\}$ indexes the elements of P and weights on the edges in set E are distances between the pairs.

We note that this problem formulation has the implicit assumption that the two robots start traveling from one pair at the same time toward the next pair and they leave the next pair only when both of them have finished processing, i.e., one robot cannot travel while the other is processing. This ensures that the two robots travel while satisfying the geometric constraints if they travel between the points with the same velocity. The cost obtained is thus an upper bound on the travel cost (as defined in Section III) and can be improved using more sophisticated trajectory optimization schemes. The TSP is an NP-hard problem and, in general, it is not even possible to get a solution within a constant factor of the optimal solution [9]. However, in our case the distance metric is symmetric and satisfies the triangle inequality. For this case, there are polynomial time heuristics some of which guarantee a solution within a constant factor of the optimal solution. A few popular heuristics for solving the TSP [10] are (a) Nearest Neighbour heuristic (b) Insertion heuristics (c) Minimum Spanning Tree (MST) heuristic (d) Christofides' heuristic (e) Lin-Kerninghan heuristic. The heuristics (a), (b), (c) and (d) are usually used to construct a tour from scratch whereas (e) is used to improve a given tour. An alternative approach is to solve an integer program formulation of the TSP with a cutting plane method [5]. However, these methods tend to be more computationally expensive. The practical algorithms for TSP with triangle inequality use a combination of these methods to solve the problem. In this paper, we use the TSP solver Concorde [1] to solve the PTSP, which has implementations of the above heuristics as well as the cutting plane method. For the results in this paper, we used the Quick-Boruvka heuristic to compute a tour from scratch and the chained Lin-Kerninghan heuristic to improve the tour. We have observed from our computational experiments that when using a different heuristic (say nearest neighbor heuristic) to compute the initial tour even though the initial tour lengths may be different the improved tour lengths did not have any significant differences.

B. Order Improvement Heuristic

As discussed before, even if we get an optimal tour of the PTSP, the optimality of the solution is with respect to the chosen compatible pairs and it may be possible to improve the tour length by changing the point pairings. We have observed that the individual TSP tours induced by the PTSP tour contain self-crossings (i.e., the tour intersects itself). We note that although the Lin-Kerninghan tour improvement heuristic is intended to remove such crossings, in our problem it does so in the pair space. Therefore, we can further improve the PTSP tour by removing the self-crossings in the individual TSP tours of the robots provided the constraints are satisfied. This removal of self-crossings is equivalent to changing the pairing among the points. Figure 5 shows a simple example of a crossing in the TSP tour of robot 2. The initial pairings are given by $(1, a), (2, b), (3, c), (4, d)$. If the pairings $(2, c)$ and $(3, b)$ are feasible then we obtain a new tour in the pair space given by $\{\ldots, (1, a), (2, c), \ldots, (3, b), (4, d), \ldots\}$. Thus the removal of self-crossings in $\mathbb{R}^2$ correspond to changing the pairings among the points.

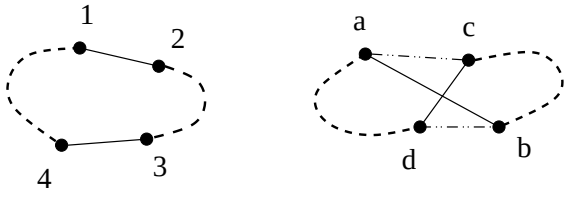

Fig. 5. The left figure shows the TSP tour of robot 1 whereas the right figure shows the self crossings observed in the TSP tour of robot 2. The initial pairings were (1,a), (2,b), (3,c), (4,d) whereas the new pairing in the lower cost tour is (1,a), (2,c), (3,b), (4,d), provided the new pairs are compatible.

Note that for a TSP in $\mathbb{R}^2$, when using the max metric, removal of a self-crossing is a neccessary but not sufficient condition for improving the tour cost (whereas for Euclidean metric it is both a neccessary and sufficient condition). Furthermore, in our PTSP formulation, the cost between two consecutive points is determined by one of the two robots. Thus, if the crossing occurs in the tour of the other robot we will not improve the overall tour cost by removing it, although the tour cost of the individual robot may improve. For example, if the cost between the pairs $(1, a)$ and $(2, b)$ given by Equation 4, is determined by the distance between points 1 and 2 then removing the self-crossing doesn't improve the overall tour cost although it may improve the individual tour cost of robot 2.

We implemented this order improvement heuristic in C++ where we consider two sets of two consecutive pairs. The number of pairs between the two sets, $i.e.$, between $(2, b)$ and $(3, c)$ in Figure 5 is a parameter (say k_c) that we can set. We first use the heuristic on the tour of robot 2 and then use it on the tour of robot 1; reversing the order did not give any significant difference in results. Table II shows the results of solving the TSP in the pair space. We computed the initial tour using the Quick-Boruvka method and then used chained Lin-Kerninghan heuristic to improve the tour. For both of these we used the implementations available in the TSP solver Concorde [1]. However, the distance function for this multi-dimensional TSP is not available within Concorde and we had to implement our own distance function. The running times were less than 400 seconds for all the cases. For the tour improvement heuristic, we chose the parameter $k_c = 5000$. The running time is dependent on the value of k_c and we did not see any substantial improvements in the tour cost with higher k_c. We observe an improvement of 5% to 8% in the tour cost using our tour improvement heuristic on the tested

datasets and the running times are less than 600 seconds. All the run times are obtained on a IBM T43p laptop (2.0 GHz processor, 1GB RAM). The final improved tour cost is given in the last column of Table II.

TABLE II
TSP TOUR OBTAINED IN PAIR SPACE WITH IMPROVED COST GIVEN BY THE ORDER IMPROVEMENT HEURISTIC

Number of pairs	Quick Boruvka Tour Cost (m)	Lin Kerninghan Tour Cost(m)	Improved Tour Cost (m)
67649	124.428	110.343	103.289
83739	140.898	124.091	115.462
90866	121.523	106.434	100.253
99729	165.527	148.353	140.162
105845	150.255	132.048	121.911

C. Singleton Insertion Heuristic

We incorporate the singleton points for each robot in its individual tour induced by the PTSP tour using a cheapest insertion heuristic, i.e., we insert a point in the tour so that the total increase in the tour cost is minimum. Let $i = (i_1, i_2)$ and $j = (j_1, j_2)$ be two consecutive pairs where the subscript 1 denotes that the point is to be processed by robot 1 and subscript 2 is for robot 2. Let k_1 be a point to be inserted in the tour of robot 1. Suppose that we want to insert k_1 between i_1 and j_1. If (k_1, i_2) do not form a compatible pair, we find the minimum distance move to be made by robot 2 to a point compatible with k_1. Otherwise, the second robot may stay at the same place. Let k_2 be the point at which we have the robot 2 when robot 1 is at k_1. The increase in the cost of the tour due to the insertion of point k_1 between i_1 and j_1 is then $C_{i_1k_1} + \max(C_{k_1j_1}, C_{k_2,j_2}) - \max(C_{i_1j_1}, C_{i_2j_2})$ where C_{pq} denotes the distance given by max metric between points p and q. We want to insert k_1 such that this tour cost increase is minimum. For each singleton, this is a linear time algorithm.

D. Evaluation of the Algorithm

In Section III we presented the path planning problem formulation wherein we proposed to divide the problem into two subproblems, namely, the splitting problem and the ordering problem. Thereafter, in the next two sections we provided algorithms for solving the two problems and provided simulation results showing the performance of the individual algorithms on example datasets. In this subsection we attempt to answer the following question: How good is the overall algorithm? The ideal answer to this question is a theoretical bound on the ratio of the cost of optimal solution to the cost of the solution of this algorithm. However, we currently do not have any such guarantees. An alternative approach, especially useful to practitioners, is to compute the performance gain achieved in using a K-robot system over a single robot system. Here we provide simulation results that show the performance gain achieved in using a two-robot system over a single robot system.

The cost of the path for any robot is the sum of the travel cost (T_t) to visit the points and the processing cost (T_p) of processing the points on the path (which is proportional to the number of points, assuming constant processing time). However, the measurement unit of the travel cost is that of length and the unit of processing cost is time in our problem formulation. So we need a weight factor between the two costs that depends on the relative importance of the two costs. We can define the performance gain obtained in using a K-robot system over a single robot system as:

$$\text{Performance Gain} = w\frac{T_p^{(1)}}{T_p^{(K)}} + (1-w)\frac{T_t^{(1)}}{T_t^{(K)}} \qquad (5)$$

where w is the weight factor ($1 \geq w \geq 0$) and the superscript denotes the number of robots used.

Table III shows the performance gain achieved by using a two robot system over a single robot system using the algorithm described in this paper. We have assumed that $w = 0.5$ or the total travel time and total processing time are equally weighted. The processing cost for the single robot is proportional to the number of points whereas the processing cost of the two robot system (given in the second column of Table III) is proportional to the sum of the number of pairs plus singletons (given in Table I). The ratio of these two costs, $i.e.$, $\frac{T_p^{(1)}}{T_p^{(2)}}$ is nearly 2. The third column in Table III gives the travel cost of the two robot system whereas the fourth column gives the travel cost of the single robot system (obtained by using the Concorde implementation of the Lin-Kernighan heuristic on a Quick-Boruvka initial tour [1]). The travel cost of the two robot system is slightly higher than the travel cost of the single robot system. We believe that this is due to a combination of the following facts: (a) we have designed the whole algorithm on first minimizing the total processing cost while satisfying the constraints and then minimizing the travel cost while keeping the processing cost fixed, (b) we have an implicit assumption that one robot cannot travel while the other is processing, which can make one robot wait even when it can move. However, as the last column demonstrates, we still have a significant gain in the overall performance. In light of the point (b) above, we can also view this gain as a lower bound on the performance gain. Note that the performance gain is also dependent on the value of w. If $w \approx 1$, $i.e.$, the processing cost dominates the travel cost, our algorithm performs very well. On the other hand, if $w \approx 0$, $i.e.$, the travel cost dominates the processing cost, the algorithm performs poorly.

TABLE III
OVERALL PERFORMANCE GAIN ACHIEVED BY USING A 2-ROBOT SYSTEM OVER A SINGLE ROBOT SYSTEM. SEE TEXT FOR DETAILS.

Num. of points	Processing Cost	2-Robot Travel Cost (m)	1-Robot Travel Cost (m)	Performance Gain
135300	67651	103.289	93.535	1.453
167536	83797	115.462	97.329	1.42
181758	90892	100.253	86.828	1.43
198570	99291	140.162	125.365	1.447
211856	106011	121.911	98.219	1.40

VI. EXTENSION TO K-ROBOT SYSTEMS

In this section we extend our splitting and ordering algorithms for the two robot system to a K-robot system. For a K-robot system the splitting problem can be set up as partitioning

the compatibility graph of the points into the minimum number of cliques of size less than or equal to K. This is a modification of the NP-hard clique partitioning problem [9], the difference being the presence of a bound on the maximum size of the clique in our case. Consequently, it is unlikely that there is a polynomial time optimal solution to this problem. Therefore we present here the extension of the greedy algorithm to the K-robot case.

A. Splitting Algorithm

We first define the K-robot splitting problem formally. For concreteness, we use our motivating problem in Figure 1 to represent the geometric constraints. We assume that the architecture is such that the K robots are mounted on the gantry in Figure 1 and each robot has independent actuation along the x-axis. The problem statement for the splitting problem is thus:

Problem Statement: Let $S = \{\mathbf{p}_j\} = \{(x_j, y_j)\}, j = 1 \ldots N$, be a set of points in $\mathbb{R}^2$. Let $Q = \{Q_i\} = \{[(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_L, y_L)]_i\}, i = 1, \ldots, N_q$, be a *ordered* set of $L-$tuples, $L \leq K$ such that $Q_i \cap Q_j = \phi$, each point in S is present in a tuple in Q, and the points in each element of Q respect the following constraints

$$\begin{aligned} & x_{li} - x_{mi} \geq s_{\min} - 2\Delta, \quad l, m \in \{1, \ldots, L\}, \quad l > m \\ & \max_{k=1\ldots L}\{|y_{ki} - \frac{1}{L}\sum_{k=1}^{K} y_{ki}|\} \leq \Delta \end{aligned} \tag{6}$$

Find such a Q of minimum cardinality.

The constraints on the x-coordinates ensure collision avoidance and the constraints on the y-coordinates indicate that the robots are constrained to move along the x-axis but have a square footprint for processing. The greedy algorithm for the 2-robot case can be extended to the K-robot case in a straightforward fashion. The main steps of the algorithm are as follows:

- Sort the points according to their y-coordinates and divide the points into bands of height 2Δ.
- For each band divide the points into almost equal K zones using K medians of the band. Number the zones 1 to K from left to right.
- Then starting with the leftmost available point in zone 1 at the first position in each tuple, assign a point to the kth position in the tuple if there is a point in zone k that satisfies the constraints, where $1 < k \leq K$. If there is more than one such point, choose the one with minimum x-coordinate breaking ties with the minimum y-coordinate. If there are no such points, then add a virtual point at the kth position.
- Repeat until all points are assigned.

The performance of this algorithm on the example datasets, shown in Table IV, shows that we obtain a very good splitting of the points among the robots. The datasets used are the same as that for the two robot case, and $s_{\min}$ and Δ are 96 mm and 8 mm respectively.

TABLE IV
GREEDY ALGORITHM RESULTS FOR FOUR-HEAD MACHINE ON EXAMPLE DATA FILES

Num. of points	Num. of quads	Num. of triples	Num. of pairs	Num. of singles	Total num. of tuples
167536	41874	5	10	5	41894
198570	49632	6	9	6	49653
211856	52954	7	5	9	52975
135300	33792	29	8	29	33858
181758	45398	5	57	37	45497

B. Ordering Algorithm

The set Q obtained as a result of the splitting problem consists of point tuples of size less than or equal to K. We set up the ordering problem as a multi-dimensional TSP in the K-tuple space, similar to the two robot case. Thereafter, we use the tour improvement heuristic to improve the tour by interchanging the points in the K-tuple space. The tuples of size less than or equal to K are then inserted into the tours of the robots using the cheapest insertion heuristic described previously. The distance between two K-tuples, say i and j, is a generalization of the definition for $K = 2$ and is defined as

$$\max\{|x_{ik} - x_{jk}|, |y_{ik} - y_{jk}|\}_{k=1}^{K} \tag{7}$$

where (x_{ik}, y_{ik}), (x_{jk}, y_{jk}) are the the coordinates of the points in the kth positions of the K-tuples i and j respectively. The tour improvement heuristic and the heuristic for inserting tuples of size less than K are direct extensions of the heuristics for the two robot system. For improving the tour cost while not changing the processing cost, we first identify the self-crossings in the tours of an individual robot and then try to interchange the order of the points assigned to that robot to achieve reduction in travel cost. For incorporating a tuple with l elements ($l < K$), we compute the insertion costs in the l-tuple space. The same concept and formula as used in Section V-C is used for determining the order of the tuples in the tour.

Table V shows the processing cost, the travel cost, and the overall performance gain achieved by using a four robot system over a single robot system. The second and third columns give the processing and travel cost respectively for the four robot system. The definition of performance gain is the same as given in Section V-D. Similar to the two robot case we see here that there is a substantial gain in the processing cost (the ratio of the second column to the first column in Table V is almost 4 for each case). However, the travel cost is worse in some cases (the ratio of travel cost in the third column and second, fourth and fifth rows of Table V to the travel cost in forth column and same rows is greater than 1) although we have an overall performance gain in all cases. Note that this performance gain is a conservative estimate and can be thought of as a lower bound on the performance gain that can be achieved.

VII. CONCLUSION

In this paper we presented algorithms for path planning of a constrained K-robot system to cover a set of points in

TABLE V
OVERALL PERFORMANCE GAIN ACHIEVED BY USING A 4-ROBOT SYSTEM OVER A SINGLE ROBOT SYSTEM

Num. of points	Processing Cost	4-Robot Travel Cost (m)	1-Robot Travel Cost (m)	Performance Gain
135300	33858	31.482	93.535	3.48
167536	41894	136.691	97.329	2.35
181758	45497	54.079	86.828	2.8
198570	49653	159.197	125.365	2.39
211856	52975	147.153	98.219	2.33

the plane. There are two relative degrees of freedom between the robots and the points, and each robot can process one point at a time within its processing footprint. The processing times for each point are assumed to be identical. We divided the path planning problem into two subproblems of splitting (and assigning) the points to each robot, and then determining an order of processing the points so that the overall tour cost is minimized. We showed that for two robots, the splitting problem can be solved optimally by converting it into a Maximum Cardinality Matching problem on a graph. However, the matching algorithm is too slow for large datasets and we developed a suboptimal $O(N \log N)$ greedy algorithm that exploits the geometric structure of our problem. For the ordering problem we first formulate and solve a TSP in the pair space (PTSP). We then improve the solution of the PTSP by identifying necessary conditions (self-crossings) for tour improvement on the (PTSP induced) tours of the individual robots. We also give a cheapest insertion heuristic on the pair space to incorporate the singletons in the ordering algorithm. We provide computational results showing the performance gain achieved by a two robot system using our algorithm over a single robot system. Finally, we generalize our splitting and ordering algorithms to K-robot systems and provide computational results showing their performance on typical industrial datasets for $K = 4$.

The division of the overall path planning problem into a splitting problem and ordering problem makes the problem tractable. However, this approach can lead to a suboptimal solution. An important question for the future is to obtain theoretical bounds on the worst-case performance of this solution procedure. In ongoing work, we are exploring extensions to systems where the processing time at each point may be different.

REFERENCES

[1] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

[2] M. Bonert, L. H. Shu, and B. Benhabib. Motion planning for multirobot assembly systems. In *Proceedings of the ASME Design engineering technical Conference*, Las Vegas, NV, September 1999.

[3] 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. *International Journal of Robotics Research*, 17(8):858–867, August 1998.

[4] N. Chakraborty, S. Akella, and J. T. Wen. Coverage of a planar point set with multiple constrained robots. In *IEEE International Conference on Automation Science and Engineering*, Scottsdale, AZ, September 2007.

[5] G. B. Dantzig, R. Fulkerson, and S. M. Johnson. Solution of a large-scale traveling salesman problem. *Operations Research*, 2:393–410, 1954.

[6] S. Dubowsky and T. D. Blubaugh. Planning time-optimal robotic manipulator motions and work places for point-to-point tasks. *IEEE Transactions on Robotics and Automation*, 5(3):377–381, March 1989.

[7] J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards*, 69B:125–130, 1965.

[8] G. N. Frederickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, May 1978.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[10] G. Gutin and A. Punnen. *Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishers, 2002.

[11] A. B. Kahng, G. Robins, and E. Walkup. New results and algorithms for MCM substrate testing. In *Proc. IEEE Intl. Symp. on Circuits and Systems*, pages 1113–1116, San Diego, CA, May 1992.

[12] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehert and Winston, New York, 1976.

[13] S. Micali and V. V. Vazirani. An $O(\sqrt{(|V|)}|E|)$ algorithm for finding maximum matching in general graphs. In *Proc. Twenty-first Annual Symposium on the Foundations of Computer Science*, pages 17–27, Long Beach, California, 1980.

[14] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall Inc., Englewood Cliffs, NJ, 1982.

[15] M. Saha, T. Roughgarden, J.-C. Latombe, and G. Sanchez-Ante. Planning tours of robotic arms among partitioned goals. *International Journal of Robotics Research*, 25(3):207–223, 2006.

[16] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost graph library: User guide and reference manual*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[17] S. N. Spitz and A. A. G. Requicha. Multiple goals with planning for coordinate measuring machines. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 2322–2327, Santa Barbara, CA, April 2000.

[18] I. Vasilescu, P. Corke, M. Dunbabin, K. Kotay, and D. Rus. Data collection, storage, and retrieval with an underwater sensor network. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys 2005)*, San Diego, CA, November 2005.

[19] D. E. D. Vinkemeier and S. Hougardy. A linear-time approximation algorithm for weighted matchings in graphs. *ACM Transactions on Algorithms*, 1(1):107–122, 2005.

[20] O. von Stryk and M. Glocker. Numerical mixed-integer optimal control and motorized traveling salesman problems. *European Journal of Control*, 35(4):519–533, 2001.

[21] C. Wurll and D. Henrich. Point-to-point and multi-goal path planning for industrial robots. *Journal of Robotic Systems*, 18(8):445–461, 2001.

[22] S.-Z. Yao, N.-C. Chou, C.-K. Cheng, and T. C. Hu. A multi-probe approach for MCM substrate testing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(1):110–121, January 1994.

APPENDIX A

We present here the mixed integer optimal control formulation of the minimum time multiple robot point set coverage problem. Although we do not use this general problem formulation in our paper, we include it here for completeness. Let there be K robots that have to visit a point set $S = \{\mathbf{p}_i\}$, where they have to perform some task that takes time $\tau_i, i = 1, \ldots, N$. Let $\mathbf{q}_k(t)$ be the state of the kth robot at time t and $\dot{\mathbf{q}}_k = f_k(\mathbf{q}_k, \mathbf{u}_k)$ be the state update equation, where $\mathbf{u}_k$ is the time dependent control input. Let

$$\omega_{ijk} = \begin{cases} 1 & \text{if robot } k \text{ visits point } i \text{ at time } t_j, \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

where $t_j, j = 1, \ldots, N$, are the (unknown) times when the robots reach a point in S. The mixed integer optimal control

problem is given by

$$\begin{aligned}
&\text{Minimize} \quad t_f \\
&\text{s.t.:} \quad \dot{\mathbf{q}}_k = f_k(\mathbf{q}_k, \mathbf{u}_k), \quad k = 1, \ldots, K && (9)\\
&\sum_{j=1}^{N}\sum_{k=1}^{K} \omega_{ijk}\mathbf{q}_k(t_j) = \mathbf{p}_i, \quad i = 1, \ldots, N && (10)\\
&\sum_{j=1}^{N}\sum_{k=1}^{K} \omega_{ijk}\mathbf{q}_k(t_j + \alpha\tau_i) = \mathbf{p}_i, && (11)\\
&\forall\alpha, \quad 0 < \alpha \leq 1, \quad i = 1, \ldots, N \\
&\sum_{j=1}^{N}\sum_{k=1}^{K} \omega_{ijk} = 1, \quad i = 1, \ldots, N && (12)\\
&\sum_{i=1}^{N}\sum_{j=1}^{N} \omega_{ijk} = 1, \quad k = 1, \ldots, K && (13)\\
&\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{K} \omega_{ijk} = N && (14)\\
&\sum_{i=1}^{N}\sum_{k=1}^{K} \omega_{ijk} \leq K, \quad j = 1, \ldots, N && (15)\\
&\mathbf{h}(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_K) \leq \mathbf{0} && (16)\\
&\mathbf{g}(\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_K) \leq \mathbf{0} && (17)\\
&t_j \leq t_f, \quad j = 1, \ldots, N && (18)
\end{aligned}$$

where $\mathbf{h}(.)$ and $\mathbf{g}(.)$ are vector-valued vector functions representing constraints on the states ($e.g.$, geometric constraints like inter-robot collision avoidance) and control inputs respectively. In addition, we can also include any constraints on the initial and final states of each robot. Equation 9 states that the state evolution of the robots should obey the dynamics constraints. Equation 10 states that for each point p_i there is exactly one robot k that visits the point at some time, say t_j, and Equation 11 states that the robot has to stay at the point i for time τ_i to complete the task. Equation 12 states that each point should be visited only once and Equation 13 implies that a robot can only be at one particular point at a time. Equation 14 states that the total number of points that have to covered by all the robots is N whereas Equation 15 implies that at a given time t_j, at most K points can be processed.