Robot Path Integration in Manufacturing Processes: Genetic Algorithm Versus Ant Colony Optimization

Girma S. Tewolde, *Student Member, IEEE*, and Weihua Sheng, *Member, IEEE*

***Abstract*—Tool path planning for automated manufacturing processes is a computationally complex task. This paper addresses the problem of tool path integration in the context of spray-forming processes. Tool paths for geometry-complicated parts are generated by partitioning them into individual freeform surfaces, generating the paths for each partition, and then, finally, interconnecting the paths from the different patches so as to minimize the overall path length. We model the problem as a variant of the rural postman problem (RPP), which we call open-RPP. In this paper, we present two different solutions to the open-RPP. The first solution is based on genetic algorithms and the second one is based on ant colony optimization. This paper presents and compares the results from both methods on sample data and on real-world automotive body parts. We conclude this paper with remarks about the effectiveness of our implementations and the pros and cons of each method.**

***Index Terms*—Ant colony optimization (ACO), genetic algorithm (GA), path integration, spray forming, tool path planning.**

I. Introduction

A. Motivation

SPRAY forming is an emerging preforming process used in the manufacturing industry to fabricate products such as body parts of automobiles and airplanes. The process of spray forming is as follows. First, glass fibers are sprayed on a mold using a chopper gun; at the same time, the binders are applied; then, a consolidation process is implemented to melt the binders so that the glass fibers are combined; finally, the finished part is obtained by removing the mold. A typical spray-forming workcell is shown in Fig. 1. Spray forming has many advantages over traditional part manufacturing processes, including lower product weight, lower material cost, and higher flexibility.

The spray-forming process is usually carried out by moving mechanisms such as robotic manipulators with specific spraying tools mounted on their end-effectors. Compared to the full automation in the spraying process, the automation in the spraying tool planning is far from satisfying. Currently, tool planning is done manually through a teaching method, which heavily relies on an operator's experience and knowledge. It usually requires an operator to use a trial-and-error approach to find a good tool plan. For example, in Ford Motor Company's Aston Martin plant, it takes an experienced operator about eight weeks to design a tool trajectory in order to spray a door panel.

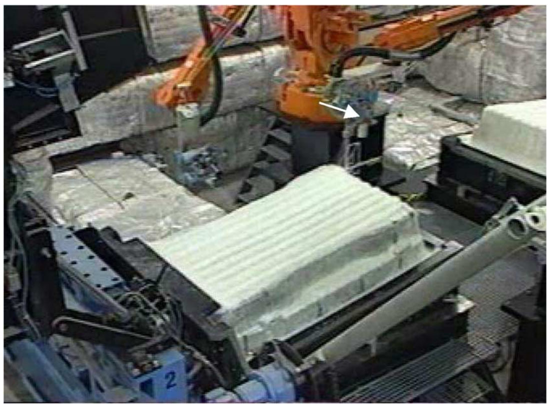


Fig. 1. Typical spray-forming workcell.

Manuscript received May 2, 2006; revised December 11, 2006. This paper was recommended by Associate Editor Kamel.

G. S. Tewolde is with the Department of Electrical and Computer Engineering, Kettering University, Flint, MI 48504 USA (e-mail: gtewolde@kettering.edu).

W. Sheng is with the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK 74078 USA (e-mail: weihua.sheng@okstate.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCA.2007.914769

B. Related Work

Robot path planning [1], [2] has been attracting many researchers over the last few decades. In recent years, there are some reports on automated tool planning for specific surface manufacturing applications. Some of the work considers the coverage problem, which only requires that any point on the surface be covered or touched by the tool. For example, Huang [3] presented an optimal line-sweep-based decomposition method to cover a surface by minimizing the number of turns a tool has to make. However, his method cannot handle freeform surfaces. Path planning for polishing robots has been investigated by several researchers. For example, path generation using the computer-aided design (CAD) model of a surface has been developed by Mizugaki *et al.* [4] and Takeuchi *et al.* [5]. However, since both methods use parametric surfaces to represent part geometry, it is very hard to handle surfaces with multiple patches or surfaces with complicated topology. The thickness problem arises from spray-painting applications, and it is more complicated than the coverage problem. Asakawa and Takeuchi [6] developed a teachingless path generation method to paint a car bumper using the parametric surfaces. However, the resulted paint uniformity is not satisfying, and no report on how to find the spray overlap percentage and the gun speed is available. Antonio *et al.* [7] developed a framework for optimal trajectory planning to deal with the optimal paint thickness problem; but in their method, the paint-gun path must be given *a priori*. Recently, Atkar *et al.* [8] presented an optimal path-planning method for a

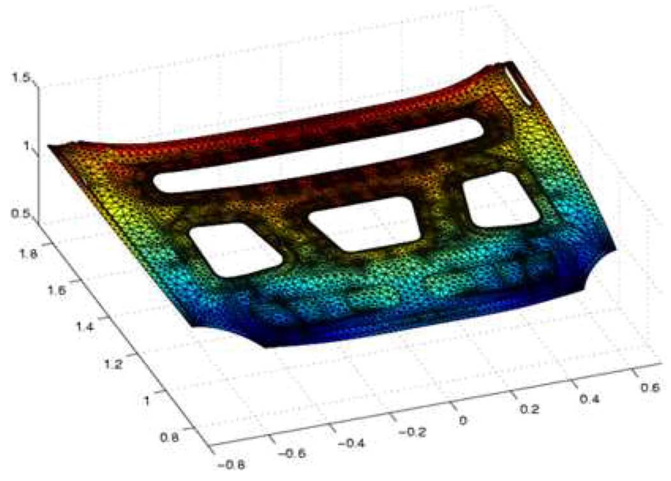


Fig. 2. Car inner hood.

single freeform surface, and their main concern is to choose the start curve so that the thickness uniformity can be guaranteed. Compared to spray painting, spray forming has received even less attention. Penin *et al.* [9] developed an automatic path-planning method to spray glass fibers on a panel with cement. The spray width and spraying speed are determined using spray rules. However, the material-thickness constraints are not considered.

To summarize, several problems need to be solved before an automated tool planning system can be adopted for factory-floor use.

1) Most of the existing tool-planning approaches can only handle path planning on a 2-D plane. Automated tool planning for freeform surfaces in 3-D space is still an open problem.
2) Most of the existing approaches can only deal with parts that have simple geometry while real-world parts usually have complicated shape or topology. A typical example is a car inner hood, as shown in Fig. 2. The complexity of the part geometry requires that the surfaces be partitioned.

This paper is organized as follows. In Section II, the general framework of tool planning is introduced. Section III discusses the tool path planning. Section III also presents the path integration problem and modeling of the problem as a variant of the rural-postman problem (RPP). Sections IV and V describe, respectively, the genetic algorithm (GA)-based and the ant colony optimization (ACO)-based algorithms developed in this paper to solve the path integration problem. Section VI presents the implementation details and results of experimental testing for both algorithms and a comparison between them. Finally, this paper concludes in Section VII.

II. General Framework

Fig. 3 shows the proposed automated tool-planning system for spray forming. There are four major inputs to this planning system, which are as follows: part CAD model, tool model, task constraints, and optimization criteria. The planning system generates a tool plan which can be validated by a simulation module and a deposition quality verification module.

A CAD model contains the geometric information of a part. A tessellation representation is used to model the part surfaces.

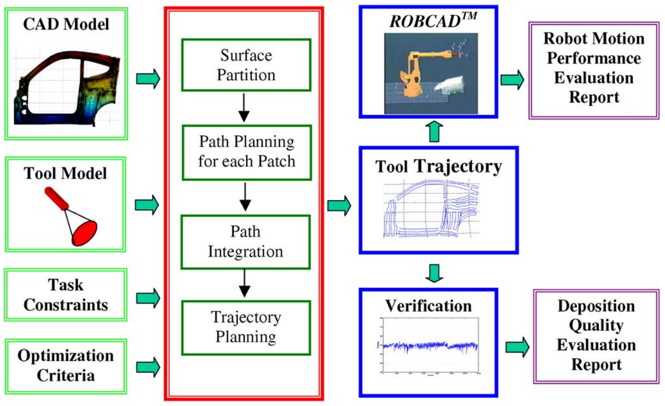


Fig. 3. Automated tool planning system for spray forming.

A tool is the deposition equipment mounted on the end-effector of a moving mechanism, such as a robotic manipulator. A general deposition tool model in spray forming can be represented by a spray cone [10]. Task constraints include thickness requirements, kinematics constraints of the moving mechanism, etc. Optimization criteria can be time, motion performance of the moving mechanism and wastage, etc.

Therefore, the automated tool-planning problem for spray forming can be formulated as follows. Given the CAD model M_c of the part surface, tool model M_t, task constraints Ω, and optimization criteria O, find a mapping $F : (M_\mathrm{c}, M_\mathrm{t}, \Omega, O) \rightarrow \mathbf{\Gamma}$. $\mathbf{\Gamma}$ is the overall tool trajectory which specifies a series of positions and associated velocities of the tool tip. In practice, a constant standoff distance h is usually used, and the tool tip is assumed to point toward the surface along the reverse of the local normal direction [7].

The overall strategy of the automated tool planning, as shown in Fig. 3, is based on a divide-and-conquer philosophy. The compound surface is first partitioned into easy-to-solve patches; second, for each patch, a path is obtained; third, the paths are integrated to form a complete path; and lastly, associated tool velocities are calculated for the complete path to obtain the overall trajectory.

III. Tool Path Planning

A. Surface Partition

The first step in the tool planning is surface partition. It is observed that the geometry of most automotive parts is complicated due to the following reasons: 1) there are holes within the outer contour and 2) the surface normals may not be consistent, and neighboring surface areas may form certain angles where they meet. These two facts introduce difficulties in planning tool trajectory that achieves high motion performance and minimum thickness error. By partitioning the surfaces, the tool-planning problem can be simplified. Based on the earlier two facts, a two-step partition process is proposed. First, the compound surface is partitioned so that each patch does not contain any hole and is simple, regular shape, or topology. Second, each patch is, if necessary, further partitioned based on the normals so that the final patches are relatively flat.

It is observed that most of the parts in surface manufacturing consist of low-curvature freeform surfaces. This observation

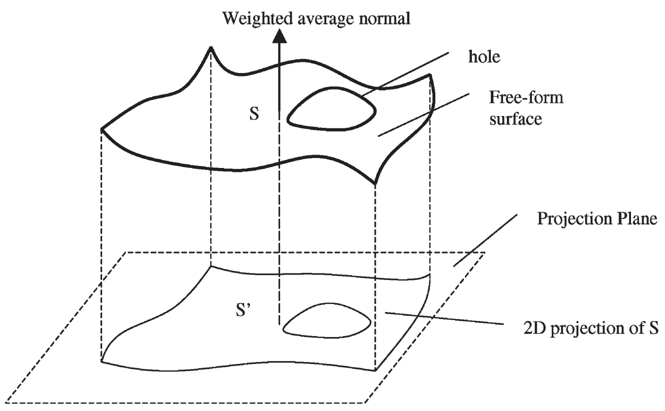


Fig. 4. Project a freeform surface to a plane.

implies that, in order to understand the topology of a freeform surface, it is sufficient to analyze the corresponding 2-D projection. The partitioning obtained in 2-D can then be mapped back to the freeform surface to realize topology-based surface partitioning (Fig. 4).

The normal of the projection plane is chosen as the weighted average normal direction [11]. It has been proved in our previous work that the projection on this direction achieves the maximum projection area [11].

Therefore, the problem of partitioning the compound surfaces is converted to the problem of partitioning a 2-D polygon into multiple subpolygons. The criterion to guide the partition is that the shape of the subpolygons should enable fast movement of the tool and uniform material deposition. Our previous work [12] proved that, among all the factors determining the speed of tool movement, the number of tool direction changes is a dominant factor because of the deceleration and acceleration processes. For the uniformity issue, we also observed that it is usually hard to achieve thickness uniformity along the common borders which receive material from paths on two different patches [10]. Therefore, we adopt the following measures to characterize a subpolygon.

1) **Regularity**: Sharp angles always bring difficulties in automated tool planning and motion control due to the small area at the corner [13], which causes many tool direction changes. Essentially, the regularity requirement implies a favor on right or obtuse interior angles.
2) **Convexity**: Concavity of a polygon also implies more changes of the tool moving direction. Therefore, it is necessary to avoid those interior angles that are greater than π in surface partitioning.
3) **Number of turns a tool makes**: Fewer turns usually imply less travelling time for the tool, since making turns requires a tool to completely stop at the ends of the paths [3]. The number of turns a tool has to make to finish a subpolygon can be represented by the minimum altitude $\mathrm{ALT}_{\min}$ of the subpolygon [3].
4) **Length of common borders**: The common border is where two subpolygons meet. It is desirable to reduce the total length of common borders to improve thickness uniformity. This polygon partition problem can be modeled as a weighted-set partition problem and solved using commercial optimization software. The detailed procedures can be found in [14].

The patches obtained through topology-based partition may exhibit big curvatures. For these patches, it is very difficult to obtain well-behaved trajectories that minimize the thickness error. Therefore, a further step is needed to partition the patches into flat patches or patches that satisfy the flatness constraint: Its maximum deviation angle (MDA) is less than a certain threshold β_{th}. A growth algorithm [11] is proposed to find flat patches, which finds the triangle with the maximum area first and then gradually pull in neighboring triangles, until the MDA exceeds β_{th}.

After the two-step surface partition, flat patches are generated. The whole compound surface can be described by a flat-patch-adjacency graph, which models each flat patch as a node and the adjacency relationship between two patches as a corresponding edge.

B. Tool Path Planning for Each Patch

The next step is to plan the tool path to cover each flat patch. Zigzag or spiral patterns are usually used to generate the paths. Zigzag patterns lead to simple tool movement, but the nonisotopic nature introduces difficulty in maintaining thickness uniformity near the patch borders. Spiral patterns have isotopic nature but may not be able to avoid disconnected path segments for certain patch shapes [15]. In most of the previous work, only one movement pattern can be preselected. In our system, the tool planner considers both patterns and uses whatever pattern is appropriate.

To determine the pattern and sweeping direction, the contour of a flat patch is partitioned into multiple border segments, based on the critical points where curvatures achieve local maximum. For simplicity, we only consider the sweeping directions parallel to border segments. There are multiple choices to design the paths. For example, Fig. 5(a)–(c) shows three zigzag patterns while Fig. 5(d) shows a spiral pattern. The pattern and sweeping direction that results in the maximum path fairness [14] will be used to generate the tool path on that patch.

With the known pattern and sweeping direction, a nominal path spacing distance d_{n} is calculated by minimizing the thickness square error on that patch. The path along the common border segments are carefully designed so that the thickness error in the neighborhood of the common border segments is minimized [10].

C. Path Integration Problem

The generated individual paths should be integrated, which means that the ends of the paths need to be connected so that the robot can transit from one path to another (Fig. 6). To achieve the minimum spray time, the integration problem can be modeled as a problem of finding the shortest travelling path for the robot. Essentially, the problem is to determine an order to traverse each individual path. This problem resembles the RPP [16]. The RPP assumes an undirected connected graph $G(V, E, R, \omega : E \mapsto Z_0^+)$, where V is the vertex set, E is the edge set, R is an arbitrary subset of E, and ω is the edge length

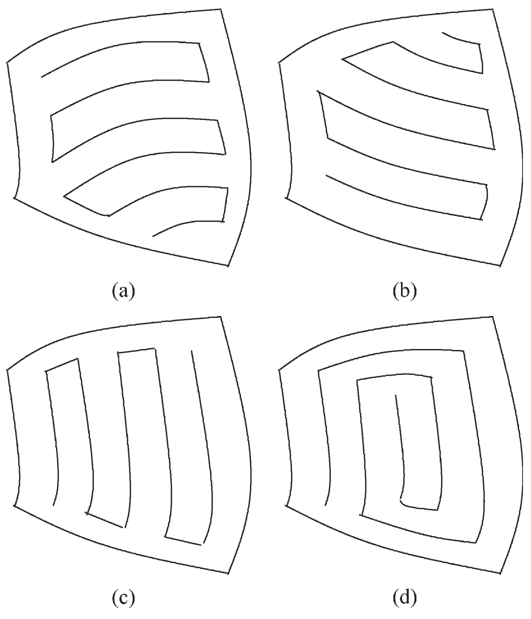


Fig. 5. Flat patch and its four sweeping directions.

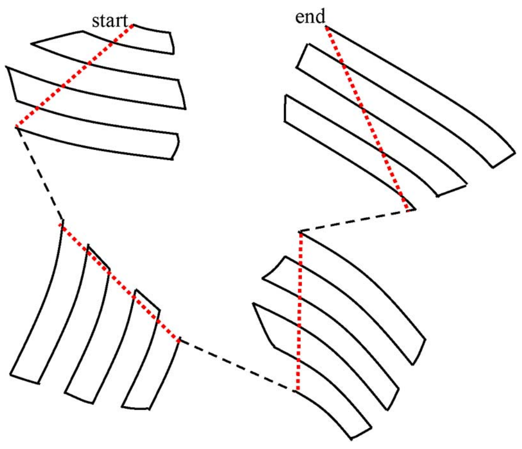


Fig. 6. Paths on different patches should be integrated with a goal to minimize the total traveling distance.

set. The objective of RPP is to find a minimum length tour in G that includes each edge of R at least once.

The classical RPP is a tour that traverses the set of all required edges and possibly some more additional edges to form a closed cycle of minimum weight. Since, for our spray-painting problem, we do not really require a closed cycle, we deal with a modified version of the RPP, that we call open-RPP. In modeling our problem, we replace the path in each patch by an edge, whose end points coincide with the end points of the tool path in that patch, and assign the direct distance between the end points as its weight. Thus, our objective is to find an optimal or near-optimal path that starts at some vertex and traverses through all the required edges.

The RPP (and, hence, the open-RPP) is a variant of the travelling salesman problem (TSP) and is NP-hard [17]; hence, there is no known polynomial-time algorithm that provides an optimum solution to the problem. Instead, many researchers resort to approximations or some heuristic methods to obtain near-optimal solutions. The solution provides a list of vertices

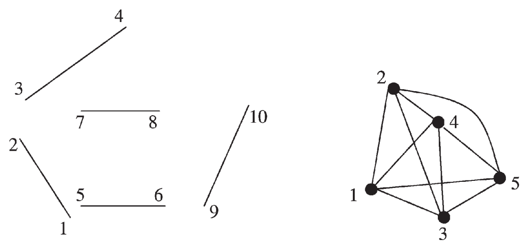


Fig. 7. Transformation of (a) an input graph to (b) a completely connected Hamiltonian graph $G^{\mathrm{H}} = (V^{\mathrm{H}}, E^{\mathrm{H}})$.

for the visiting order of the tool to tour all the assigned paths, with a minimal total cost.

In this paper, we develop two methods to solve the open-RPP problem: the first is a GA-based method, and the second one is based on the ACO method. Both of these and other nature-inspired methods are becoming increasingly popular in solving hard optimization problems [16], [18]–[23]. The next two sections describe the implementation details of the two algorithms developed in this paper to generate near-optimal solutions to the path integration problem.

IV. GA to Solve the Open-RPP

Our implementation of the GA was inspired by the work of Kang and Han [20], who propose a graph transformation method by using a Hamiltonian graph that represents all the required edges in the original problem. On the other hand, Cook *et al.* [16] describe a different approach, using a GA for only generating a minimum-weight augmenting set of required edges. Then, they apply a basic heuristic to construct completion of the graph, using the augmented set of required edges, to obtain an Eulerian tour.

Using the graph-transformation approach, each of the required edges in the original graph is mapped into corresponding vertices, and then, a complete Hamiltonian graph, $G^{\mathrm{H}} = (V^{\mathrm{H}}, E^{\mathrm{H}})$, is formed. Therefore, our problem reduces to that of determining an optimal Hamiltonian cycle for the transformed graph whose solution gives an optimal order of visiting the required edges in the original problem. Fig. 7 shows an example of this transformation process.

The weights of the edges in G^{H} are not fixed; instead, they depend on the visiting order of the pair of vertices in each required edge and the route between the pair of edges represented by the nodes in G^{H}. For example, if we consider two partial paths, 1–2–3–4 and 2–1–4–3, between required edges 1–2 and 3–4 in Fig. 7(a), we can see that the total distance of the paths are not the same. The weights of the edges in the Hamiltonian graph, as shown in Fig. 7(b), are determined during the computation of the cost of the path obtained by each individual solution during the application of the GA. At the start of the program, we compute the shortest distances and paths between all pairs of vertices in the original graph and store the result in a matrix for use in computation of the routing costs.

The GA approach attempts to search the entire solution space to find a global optimal solution, applying randomized evolutionary operations to prevent it from being trapped in local optima. For the success of this methodology, an efficient and good representation of the solution, i.e., the chromosome structure, is essential. For most combinatorial problems, there is a constraint that the alleles in the chromosome should be unique.

1) Chromosome Structure: The vertices in V^{H} are arranged to form the alleles in the chromosome. The chromosome has a length of $|V^{\mathrm{H}}|$. The order of the vertices in the chromosome defines the order of the routing path. Since each vertex represents an edge in the original graph, we need to identify the order of traversing the edge from one end point to another, or backward. This information is encoded as a separate component of the chromosome structure and evolves alongside it. An example of the format of the chromosome structure, for $|V^{\mathrm{H}}| = 6$, is shown as follows:

$$P_{\mathrm{i}} = 3 \quad 1 \quad 2 \quad 5 \quad 4 \quad 6 \qquad \mathrm{Ps}_{\mathrm{i}} = 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0.$$

P_{i} is an individual solution in the population, listing the order of visiting the required edges in the open-RPP path. Initially, we assign the visiting directions for the required edges arbitrarily, identifying start and end vertices of the edges. As the populations of solutions in the GA evolve, the directions of visiting the required edges in the open-RPP path also evolve. This information is maintained in the Ps_{i} parameter. A value of "1" for a visiting direction in the Ps_{i} will follow the initially assigned order, while a value of "0" will follow the reverse order. The visiting order affects the cost of the solution, and we use this information in computing the fitness of individual solutions. We find this chromosome-encoding method effective, since in most cases, it converges in fewer generations than the approach that encodes only the vertex sequences (i.e., using P_{i} only that was suggested in [20]).

The process of finding a global optimum solution using GA involves the application of some randomized genetic operations that are designed to explore the entire solution space. The crossover and mutation operations used in our implementation are described as follows.

2) Crossover: The crossover operator exchanges substrings in two selected chromosomes and produces a pair of new chromosomes. Among several crossover operators that are proposed for permutation representations, we choose to apply the order crossover (OX) [18] on the P_{i}'s and the ordinary crossover operator on the Ps_{i}'s. The OX guarantees the creation of legal permutation of the vertices in the resulting offsprings. Another common crossover operator is partially mapped crossover, which Kang and Han [20] use in their paper. Fig. 8 shows the OX and the ordinary crossover operators as applied to a pair of parent solutions. A randomly chosen continuous block of alleles (the numbers shown in the boxes) are inherited as they are from the parents, while the remaining portions of the chromosomes are exchanged between the parents to generate the offsprings. In the OX method, which was applied to the P_1, P_2 pair to generate the C_1, C_2 pair, the resulting numbers are then manipulated to guarantee that the list contains the permutation of all the vertices (replacing the redundant numbers that are shown in circles by those missing from the list).

The parents for the crossover operation are selected randomly. But to increase the chances of generating better offsprings, we take five pairs of random samples of the whole population and pick the best pair of individuals based on their fitness for the crossover operation. The crossover rate determines the number of crossover operations to apply in the current generation of solutions before making selections to proceed to the next generation.

$$\begin{array}{ll}
P_1 = 3\ 1\ \boxed{2\ 5\ 4}\ 6 & \mathrm{Ps}_1 = 0\ \boxed{0\ 1\ 0}\ 1\ 0 \\
P_2 = 6\ 3\ \boxed{4\ 2\ 1}\ 5 & \mathrm{Ps}_2 = 1\ \boxed{1\ 0\ 1}\ 0\ 1 \\
 & \\
C_1' = 6\ 3\ \boxed{2\ 5\ 4}\ \text{\textcircled{5}} & \\
C_2' = 3\ \text{\textcircled{1}}\ \boxed{4\ 2\ 1}\ 6 & \\
 & \\
C_1 = 6\ 3\ \boxed{2\ 5\ 4}\ 1 & \mathrm{Cs}_1 = 1\ \boxed{0\ 1\ 0}\ 0\ 1 \\
C_2 = 3\ 5\ \boxed{4\ 2\ 1}\ 6 & \mathrm{Cs}_2 = 0\ \boxed{1\ 0\ 1}\ 1\ 0
\end{array}$$

Fig. 8. OX operator applied to parents P_1, P_2 and the ordinary crossover operator applied to parents P_{s1}, P_{s2} resulting in the offspring pairs C_1, C_2 and C_{s1}, C_{s2}, respectively.

3) Mutation: In our implementation, we apply the reciprocal-exchange mutation on the P_{i}'s by selecting two positions at random and swapping the vertex values at those positions. Mutation on the Ps_{i}'s is done by swapping two randomly selected bits and flipping the value of another randomly chosen bit. Like in the crossover operation, mutation is applied on randomly picked individuals, and the mutation rate determines the number of times the operation is repeated on the current generation of solutions before evolving to the next generation.

4) Fitness Evaluation: Selection of individual solutions to the next generation is made based on their fitness values. In our implementation, the total number of individual solutions at the end of every generation is

$$P_{\mathrm{total}} = P_{\mathrm{size}} + x_{\mathrm{rate}} * P_{\mathrm{size}}.$$

Note that, since mutation only modifies the individual on which it is applied, it does not generate a new offspring. P_{size} is the population size that the algorithm maintains, x_{rate} is the crossover rate, and P_{total} is the total population at the end of each generation. When proceeding to the next generation during the evolution process, the fitness of each individual is evaluated, and the best P_{size} individuals are chosen by the principle of the survival of the fittest.

The total cost of each open-RPP path computed as its tour length (TL) is used as a measure of the fitness value. Floyd–Warshall's [24] algorithm is applied to compute the shortest distances between all pairs of vertices in the original graph, and this information is used for the fitness computation.

V. ACO to Solve the Open-RPP

Inspiration from nature continues to provide insight into new approaches in tackling complex computational problems. The class of ACO [21] algorithms is such an example of nature's inspiration as are GA and neural networks. Dorigo *et al.* [21] proposed the use of the ant colony as cooperating agents to solve combinatorial optimization problems. This class of algorithms are versatile and robust, and they are finding applications in many static and dynamic optimization problems [22], [23].

Dorigo *et al.* [21] demonstrated their ant system algorithm by solving the classical TSP. The analogy is taken from the way ant colonies function in finding the shortest path to a food source and the sharing of information by depositing a so-called pheromone on the trail. Each individual ant is a simple creature, but the emergent complex behavior of the ant colony helps find the best solution in the search space by distributing the efforts of individual ants and sharing information by using the environment. This is a powerful technique and is applicable to numerous search and optimization problems. Sim and Sun [23] exploited the adaptive nature of this class of algorithms to tackle the dynamic-routing and load-balancing problems in networks.

The open-RPP problem that we are addressing in this paper is also a problem of combinatorial optimization with the objective of finding a tour of minimum length that covers all required edges in a given graph. In the remaining parts of this section, we describe the ACO-based algorithm developed in this paper.

A. Algorithm Description

In this section, we describe the general idea of the algorithm we developed to solve the open-RPP by considering the simple graph shown in Fig. 7. This graph of six required edges and 12 vertices is first transformed into a complete graph by forming edges between all pairs of vertices and by assigning the shortest path distances between the vertices (computed using Floyd–Warshall's [24] algorithm as described earlier) as the weights for the corresponding edges. Providing such shortest paths allows the ants to choose optimum paths to connect the required edges.

During the execution of the ACO-based algorithm, a number of ants are created to search for the best solution starting from each vertex of the required edges. Initially, each ant initializes a list of the vertices it has not visited yet (NVV) and another list of vertices it has already visited (VV). Similar information is also compiled about the required edges, i.e., for the not visited edges (NVE) and visited edges (VE).

For example, if ant 1 is set to start the execution of the ACO-based algorithm at vertex 1, it initializes the arrays as follows. The TL is initialized to zero at the start of the search in each cycle.

$\mathrm{NVV}[1] = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$
$\mathrm{VV}[1] = \{\}$
$\mathrm{VNE}[1] = \{(1, 2), (3, 4), (5, 6), (7, 8), (9, 10), (11, 12)\}$
$\mathrm{VE}[1] = \{\}$
$\mathrm{TL}[1] = 0.0.$

The ants in the colony need to share information about the quality of the solution they have generated during each execution cycle. Such information communication is expected to influence the ants' decisions in their choice of paths when they build an optimum open-RPP path. In effect, the individual ants in the colony explore different regions of the solution space in parallel, and they provide feedback to the colony by appropriately modifying this quantity, which is called trail value (TV), on the edges of the graph. This mechanism is believed to stir the solution toward the optimum value and to facilitate convergence.

The TV accumulated on edge (x, y) since the start of the ACO-based algorithm is designated by TV_{xy}. It is contributed by all the ants that included this edge in their solution during the previous cycles of the ACO-based algorithm. TV_{xy} values change with time (it gradually decays with time due to the evaporation process and it increases in value when ants follow the edge). The TV_{xy} value therefore tells the ants how favorable the edge (x, y) has been so far, and hence, it affects their choice of path when building a solution.

The amount by which an individual ant contributes to the TV $(\triangle\mathrm{TV}_{xy})$ depends on the quality of the solution it generated, which is measured in terms of the open-RPP TL. Hence, longer tours contribute less than shorter tours. This is a positive feedback about past performance, and it influences the decisions of future steps of the ACO-based algorithm. Ants in the succeeding cycles will then have higher probability to choose edges with higher TVs, thus reinforcing the feedback further and improving the solution quality. We have used the formula of Dorigo *et al.* [21] in updating the TVs on each edge as follows:

For every edge $e = (v_x, v_y)$ in the graph
  Initialize $\triangle\mathrm{TV}_{xy} = 0$
For each ant $i = 1$ to n_a do

$$\triangle\mathrm{TV}_{xy} \Leftarrow \triangle\mathrm{TV}_{xy} + \frac{Q}{\mathrm{TL}[i]}. \tag{1}$$

then compute TV_{xy} using

$$\mathrm{TV}_{xy} \Leftarrow \rho \cdot \mathrm{TV}_{xy} + \triangle\mathrm{TV}_{xy}. \tag{2}$$

Equation (1) shows the aggregation of the TV contributions on edge (v_x, v_y) by all ants that include the edge in their path. The quantity Q is a constant parameter. In (2), the value of ρ (a value < 1) determines the evaporation rate of the TV. If some edge is not selected by many ants in most of the previous cycles, then its TV will decay at the rate of ρ. As the TV_{xy} value of an edge gets smaller due to the evaporation effect, it will be less likely for ants to choose the edge in their path. The second term in (2) $\triangle\mathrm{TV}_{xy}$ is the total amount of the TV deposited on an edge by all ants in the current cycle.

At every step of building the open-RPP path, an ant's decision on the choice of the next edge for the path is made based on transition probability (TP). What makes the ACO-based algorithm powerful is that the decision is based on both past experience from previous cycles (stored in the TV values) and the physical configuration of the graph, such as the distance information between pairs of vertices. Equation (3) [21] shows how the ith ant computes the TP on each of the edges from its current position (vertex x). The possible legal moves of the ant from its current position x is the list of vertices that form end points of the required edges but have not been visited yet

$$\mathrm{TP}^i_{xy} = \begin{cases} \frac{(\mathrm{TV}_{xy})^{\alpha} \cdot (\eta_{xy})^{\beta}}{\sum_{w \in \mathrm{legal}_i} (\mathrm{TV}_{xw})^{\alpha} \cdot (\eta_{xw})^{\beta}}, & \text{if } w \in \mathrm{legal}_i \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where η_{xy} is called visibility and it is computed as $1/d_{xy}$, where d_{xy} is the shortest distance between vertices v_x and v_y. α and β are parameters that control the relative importance of trail intensity versus visibility. Therefore, the TP is a tradeoff between visibility (which states that closer vertices should be chosen with high probability) and trail intensity (that states that

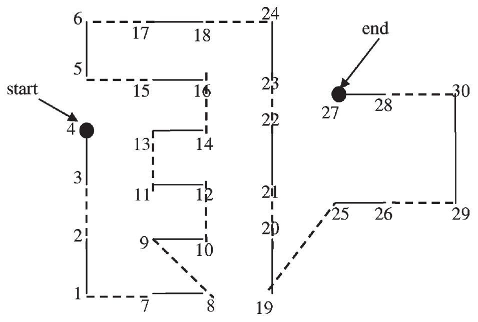


Fig. 9. Sample data and the open-RPP path generated by the GA-based algorithm.

if there has been a lot of traffic on edge (v_x, v_y), then it is highly desirable) [21].

VI. Implementation and Test Results

The two algorithms are implemented in C and run on sample data as well as on real auto parts from Ford Motor Company. This section presents the test results of the algorithms and gives a comparative discussion of the solution quality and runtime performance of the algorithms. The execution times in all tests are measured on a DELL Latitude Laptop with a 2-GHz CPU and 256-MB RAM.

A. *Test Results of the GA-Based Solution*

1) Test on Sample Data: The GA-based open-RPP algorithm was first tested on some hand-generated sample data. Fig. 9 shows the graph of a data set, containing 30 vertices and 15 required edges, and the optimal path generated by the program. Characteristics of the average and minimum cost of the solutions as the algorithm evolves over successive generations is shown in the chart in Fig. 10.

The results demonstrate that our GA implementation reaches at the best solution in fewer than 100 generations with a population size of 100, for reasonable sizes of problems. Tuning of the algorithm is done by running it with different values of population size (p_{size}), crossover rate (x_{rate}), and mutation rate (m_{rate}) to see their effect on convergence rate. Good performance was achieved for $p_{\text{size}} = 100$, $x_{\text{rate}} = 0.20$, and $m_{\text{rate}} = 0.05$. Compared to the relatively higher parameter values of Kang and Han [20] ($x_{\text{rate}} = 0.6 \sim 0.8$, $m_{\text{rate}} = 0.03 \sim 0.04$), our lower crossover rate means that we can achieve the task at a lower computational cost. In addition, in most small-to-medium-size problems, our implementation converges at the best solution with fewer than 100 generations, which is an order of magnitude lower than the 1000 generations used in [20]. Results of more test experiments on problems of increasing levels of complexity are provided in Table I and compared with the results of the ACO-based solution.

2) Test on Real Parts: We also ran the open-RPP algorithm on real parts from Ford Motor Company. For the car inner hood, as shown in Fig. 2, the optimal partitions are shown in Fig. 11. Corresponding to this partition, the generated paths shown in Fig. 12 are integrated using our GA-based algorithm to produce an optimal path, as shown in Fig. 13. Fig. 14 shows how the average and best costs of the computed open-RPP path

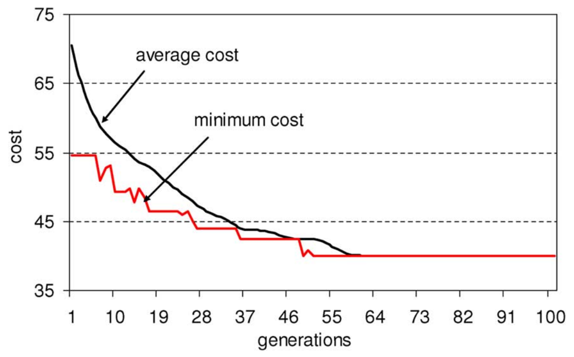


Fig. 10. Evolution of the open-RPP solution by the GA-based algorithm for the sample data shown in Fig. 9.

evolve with the number of generations. The algorithm is able to converge to the best solution in less than 50 generations, as shown in the graph. Results of the algorithm, as applied on two more automotive body parts, are presented in Table III alongside with the results of the ACO-based solution.

B. *Test Results of the ACO-Based Solution*

Like many other heuristic algorithms, the ACO-based algorithm involves a number of parameters that need to be properly tuned for good performance. The most important parameters of interest that affect the performance and quality of the solution are α, β, ρ, and Q. For best results, we used the following settings [21]: $\alpha = 1$, $\beta = 5$, $\rho = 0.5$, and $Q = 100$.

1) Test on Sample Data: The ACO-based algorithm is tested on ten sets of sample test data. The simplest one (data set #1) has only six vertices and ten edges, out of which three are designated as required edges. The rest of the data sets have increasing numbers of vertices and required edges.

The simulation records the shortest path length computed by the algorithm and its execution time. Effects of the number of ants and the number of cycles each ant executes are studied by running the simulation with different settings of these parameters. For each test case, an average value of ten simulation runs is taken.

Initially, the number of ants was set to twice as many as the number of required edges in the graph, each ant starting at one end of a required edge, and the number of ant cycles was set to 100. Such settings produce open-RPP TLs that are shorter by up to 13% than the GA-based solution. However, the ACO-based algorithm has an inherent complexity due to the following: 1) the need to compute transition probabilities; 2) the need to search for the best edge to follow at each move; and 3) the need to update the TVs after completion of each cycle. For example, the largest problem considered in the simulation requires the manipulation of 80-by-80 matrices of transition probabilities and TVs, which demands expensive computation time. The total runtime taken by the basic ACO-based algorithm in this test case was 42.9 s (which is excessively large as compared to 0.34 s taken by the GA-based solution). However, experimenting with varying numbers of cycles reveals that the ACO-based algorithm actually converges in much fewer than 100 cycles. Good quality TLs are achievable within as little as ten cycles, thus helping reduce the total execution time by over 90%.

TABLE I
RESULTS OF TESTING THE ACO-BASED AND GA-BASED ALGORITHMS ON SAMPLE TEST DATA. VALUES UNDER COLUMN ACO1 ARE RESULTS OF DEPLOYING ONE ANT PER REQUIRED EDGE, WHILE ACO2 IS FOR TWO ANTS PER REQUIRED EDGE. THE ALGORITHM RUNS TEN CYCLES IN BOTH CASES (PATH LENGTHS ARE IN METERS AND EXECUTION TIME IN SECONDS)

Data Set	Path Length (ACO1)	Path Length (ACO2)	Path Length (GA)	ACO2 vs. GA (% improvement)	Exec. Time (ACO1)	Exec. Time (ACO2)	Exec. Time (GA)
#1 (6 v, 3 re)	16.0	16.0	16.0	0.0%	< 0.01	< 0.01	0.27
#2 (10 v, 5 re)	21.12	21.12	21.12	0.0%	< 0.01	0.01	0.30
#3 (20 v, 10 re)	32.56	32.56	34.45	5.8%	0.05	0.10	0.31
#4 (30 v, 15 re)	38.53	38.53	39.06	1.3%	0.15	0.25	0.33
#5 (40 v, 20 re)	63.72	63.72	68.71	7.8%	0.30	0.51	0.35
#6 (50 v, 25 re)	68.42	67.06	73.94	10.3%	0.52	0.93	0.37
#7 (60 v, 30 re)	83.45	83.28	91.66	10.1%	0.87	1.57	0.39
#8 (64 v, 32 re)	87.11	86.87	95.66	10.1%	1.01	1.89	0.41
#9 (70 v, 35 re)	89.66	89.49	100.62	12.4%	1.29	2.41	0.45
#10 (80 v, 40 re)	102.10	102.10	115.48	13.1%	1.91	3.56	0.52

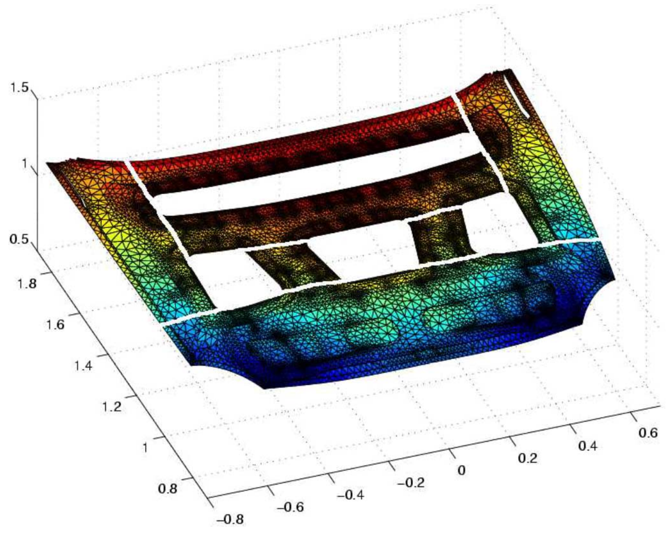


Fig. 11. Optimal partition of the car inner hood.

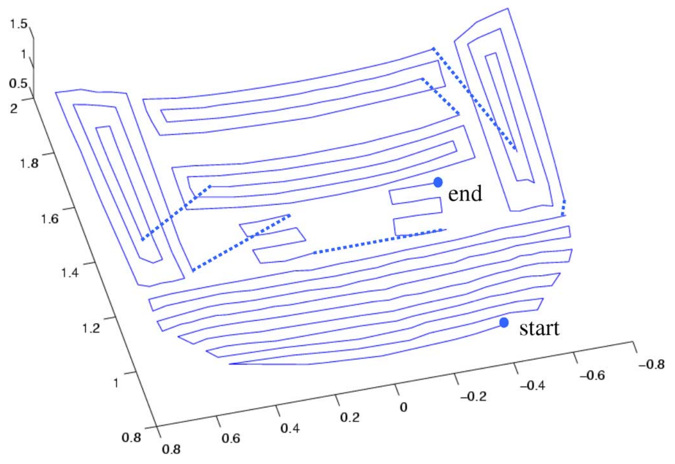


Fig. 13. Integrated path generated by the GA-based algorithm for the car inner hood.

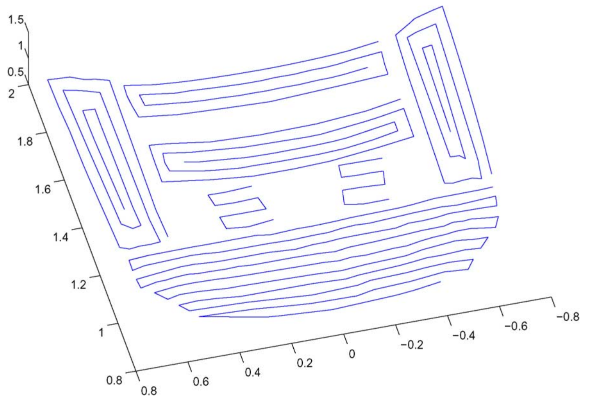


Fig. 12. Optimal paths for individual patches of the car inner hood.

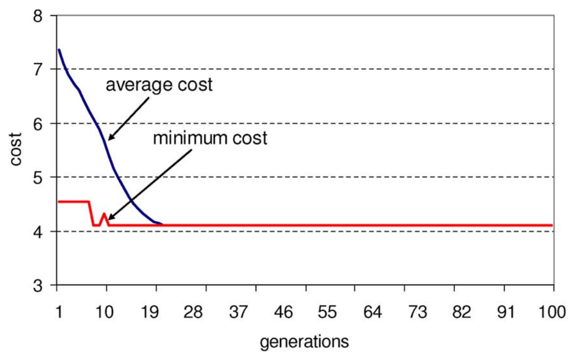


Fig. 14. Evolution of the open-RPP solution using the GA-based algorithm for the car inner hood.

Table I shows the results of further tests obtained by varying the numbers of ants searching for the optimum solution in the ACO-based algorithm. Using half as many ants (i.e., setting the number of ants to be equal to the number of required edges) has little impact on the quality of the solution, but it needed less than half the execution time as compared to the case when the number of ants is set to two times the number of required edges. In Table I, the ACO1 and ACO2 columns, respectively, refer to these two cases. The GA columns in Table I refer to the best results obtained from the GA-based solution with the following parameter settings: number of generations $= 1000$; population size $= 200$; crossover rate $= 0.2$; and mutation rate $= 0.05$.

Even further reducing the number of ants does not have much impact on the solution quality but it has significant

TABLE II
RESULTS OF TESTING THE ACO-BASED ALGORITHMS ON SAMPLE TEST DATA USING DIFFERENT NUMBERS OF ANTS. THE ALGORITHM IN EACH CASE RUNS FOR TEN CYCLES (PATH LENGTHS ARE IN METERS AND EXECUTION TIME IN SECONDS)

Data Set	3 Ants P. Length	3 Ants E. Time	5 Ants P. Length	5 Ants E. Time	10 Ants P. Length	10 Ants E. Time	ACO1 P. Length	ACO1 E. Time	ACO2 P. Length	ACO2 E. Time
#1 (6 v, 3 re)	16.0	< 0.01	16.0	< 0.01	16.0	< 0.01	16.0	< 0.01	16.0	< 0.01
#2 (10 v, 5 re)	23.12	< 0.01	21.12	< 0.01	21.12	0.01	21.12	0.01	21.12	0.01
#3 (20 v, 10 re)	32.56	0.02	32.56	0.03	32.56	0.05	32.56	0.05	32.56	0.10
#4 (30 v, 15 re)	40.8	0.04	40.8	0.06	39.06	0.11	38.53	0.15	38.53	0.25
#5 (40 v, 20 re)	64.55	0.06	65.13	0.10	63.72	0.18	63.72	0.30	63.72	0.51
#6 (50 v, 25 re)	68.84	0.10	68.84	0.15	68.84	0.25	68.42	0.52	67.06	0.93
#7 (60 v, 30 re)	83.45	0.14	83.45	0.20	83.45	0.34	83.45	0.87	83.28	1.57
#8 (64 v, 32 re)	87.45	0.16	87.45	0.23	86.87	0.36	87.11	1.01	86.87	1.89
#9 (70 v, 35 re)	98.92	0.19	89.96	0.25	89.96	0.43	89.66	1.29	89.49	2.41
#10 (80 v, 40 re)	108.58	0.24	104.8	0.32	104.64	0.55	102.10	1.91	102.10	3.56

gain in the execution-time performance of the algorithm, as shown in Table II. For example, for the largest problem size (data set #10), the best TL obtained by the algorithm using 3, 5, 10, 40, and 80 ants is, respectively, 108.58, 104.8, 104.64, 102.10, and 102.10 m, with the program execution times, respectively, of 0.24, 0.32, 0.55, 1.91, and 3.56 s in the five cases. Therefore, a small sacrifice in the solution quality (reduction by about 2.64% from the best solution, by using only 5 ants than 80 ants) gives over 90% improvement in the execution-time performance.

2) Test on Real Parts: The ACO-based algorithm is also tested on real parts from Ford Motor Company. For the generated paths on the partitions shown in Fig. 12, the algorithm produces the path integration whose result turns out to be the same as the result obtained by the GA-based solution, which is shown in Fig. 13 and in the first row of Table III. Compared to the tests performed on the large graphs of the sample data shown in the previous section, the sizes of the real parts considered in this paper were small enough to give the same results by the two variants of the ACO-based algorithm (ACO1 and ACO2). The other real test data are two different door panels whose results of the best open-RPP path obtained from the ACO-based and GA-based solutions are shown in the second and third rows of Table III. In the same table, the paths generated by these algorithms are also compared to arbitrary paths generated by a human operator.

TABLE III
RESULTS OF TESTING THE ACO-BASED AND GA-BASED ALGORITHMS ON REAL CAR PARTS (PATH LENGTHS ARE IN METERS)

Part Name	Path Length (ACO)	Path Length (GA)	Path Length (Random)
Inner Hood	4.11	4.11	7.14
Door Panel #1	20.79	22.71	29.05
Door Panel #2	34.36	35.48	46.21

C. Discussions

To summarize, for small problem sizes, the quality of the solution generated by both methods is about the same. But with increasing size and complexity of the problem, the ACO-based algorithm achieves better quality solutions as compared to the GA-based algorithm. In our experiments, the cutoff point between the two methods is at about a problem size of 20 vertices and 10 required edges. For the largest problem size considered in the experiments, the TL generated by the ACO-based algorithm is shorter by 13% than that of the GA-based algorithm.

But an interesting behavior of the GA-based implementation is that the growth rate of the execution time of the algorithm is close to flat as compared to the high growth rate exhibited by the ACO-based method. The reason is that, in the GA-based algorithm, the population size and the number of generations is set independent of the problem size. However, the number of ants used in the ACO1 and ACO2 implementations are set to grow with the number of required edges in the problem. This results in higher computational costs with increasing problem sizes. Moreover, the cycle length that each ant has to complete also increases with the size of the problem. Besides, there is a larger search space at each step when an ant is to decide on the next leg of its route as the number of choices it needs to examine based on the TVs increases. Thus, the cumulative effect of these factors makes the execution time of the ACO-based algorithm to grow faster than that of the GA-based algorithm.

However, a closer look at the effects of the number of cycles and number of ants on the quality and cost of the solution reveals that the ACO-based algorithm converges in as few as ten cycles, greatly improving the execution time. The experimental results also demonstrate a tradeoff between the solution quality and the execution time by varying the number of ants employed in the search for the optimum solution. Thus, with just a little sacrifice in the solution quality (by about 2.64% from the best solution), using as few as five ants, offers over 90% improvements in the execution-time performance.

The path integration algorithms developed in this paper have great potentials to solve many robotic path-planning problems. For example, they can be used in manufacturing applications such as robotic arc welding, painting, as well as humanitarian demining and lawn mowing, where path integration is an important step in the planning. Reducing the total path length can certainly save time and energy for these applications.

VII. CONCLUSION

This paper addresses a challenging problem in automated tool path planning for spray forming, i.e., how to design tool paths that can achieve optimized performance for compound surfaces. First, solving this problem requires the partitioning of part surfaces of complex geometries and the careful generation of tool path and trajectory of each partition. Then, the path integration problem is modeled as a variant of the RPP, which is solved using two different methods. The first method presented is based on GA, while the second one is based on the ACO. Experimental results demonstrate the effectiveness of both methods and the quality of solutions that can be achieved. We have shown a tradeoff between the quality of the solution and the cost of execution time of the ACO-based algorithm. Overall, the results show that, with increasing complexity of the problem, the ACO-based method produces better quality solutions (by up to 13% for the largest problem size we considered) than the GA-based method, at a cost of higher execution time.

Overall, our tool path planning and integration algorithms extend the traditional robot path planning into manufacturing applications, because the vast majority of research in robot-motion planning does not have the constraints that our tool path planning has to deal with, such as the uniformity, tool movement performance, etc. To further improve our tool path planner, we will investigate how to combine human skills and expertise into the path planning. In this area, some previous work exists, which uses human operator's skills to train a robot path planner [25], [26].

References

[1] J. T. Schwartz and M. Sharir, "A survey of motion planning and related geometric algorithms," *Artif. Intell.*, vol. 37, no. 1–3, pp. 157–169, Dec. 1988.

[2] M. Sharir, "Algorithmic motion planning in robotics," *Computer*, vol. 22, no. 3, pp. 9–20, Mar. 1989.

[3] W. H. Huang, "Optimal line-sweep-based decompositions for coverage algorithms," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2001, pp. 27–32.

[4] Y. Mizugaki, M. Sakamoto, and K. Kamijo, "Fractal path application in a metal mold polishing robot system," in *Proc. 5th Int. Conf. Adv. Robotics*, Pisa, Italy, Jun. 1991, pp. 431–436.

[5] Y. Takeuchi, D. Ge, and N. Asakawa, "Automated polishing process with a human-like dexterous robot," in *Proc. IEEE Int. Conf. Robot. Autom.*, Atlanta, GA, 1993, pp. 950–956.

[6] N. Asakawa and Y. Takeuchi, "Teachingless spray-painting of sculptured surface by an industrial robot," in *Proc. IEEE Int. Conf. Robot. Autom.*, Albuquerque, NM, Apr. 1997, pp. 1875–1879.

[7] J. K. Antonio, R. Ramabhadran, and T. L. Ling, "A framework for optimal trajectory planning for automated spray coating," *Int. J. Robot. Autom.*, vol. 12, no. 4, pp. 124–134, 1997.

[8] P. N. Atkar, H. Choset, and A. A. Rizzi, "Towards optimal coverage of 2-Dimensional surfaces embedded in $\mathbb{R}^3$: Choice of start curve," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2003, pp. 3581–3587.

[9] L. F. Penin, C. Balaguer, J. M. Pastor, F. J. Rodriguez, A. Barrientos, and R. Aracil, "Robotized spraying of prefabricated panels," *IEEE Robot. Autom. Mag.*, vol. 5, no. 3, pp. 18–29, Sep. 1998.

[10] H. Chen, N. Xi, Z. Wei, Y. Chen, and J. Dahl, "Robot trajectory integration for painting automotive parts with multiple patches," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2003, vol. 3, pp. 3984–3989.

[11] W. Sheng, N. Xi, M. Song, Y. Chen, and J. S. Rankin, III, "Automated CAD-guided automobile part dimensional inspection," in *Proc. Int. Conf. Robot. Autom.*, 2000, vol. 2, pp. 1157–1162.

[12] H. Chen, W. Sheng, N. Xi, M. Song, and Y. Chen, "Automated robot trajectory planning for spray painting of free-form surfaces in automotive manufacturing," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2002, vol. 1, pp. 450–455.

[13] Y. Yang, H. T. Loh, F. Y. H. Fuh, and Y. G. Wang, "Equidistant path generation for improving scanning efficiency in layered manufacturing," *Rapid Prototyping J.*, vol. 8, no. 1, pp. 30–37, 2002.

[14] W. Sheng, H. Chen, N. Xi, J. Tan, and Y. Chen, "Optimal tool path planning for compound surfaces in spray forming processes," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2004, pp. 45–50.

[15] J. H. Kao and F. B. Prinz, "Optimal motion planning for deposition in layered manufacturing," in *Proc. ASME Design Eng. Tech. Conf.*, Atlanta, GA, Sep. 1998, pp. 1–10.

[16] C. Cook, D. A. Schoenefeld, and R. L. Wainwright, "Finding rural postman tours," in *Proc. ACM Symp. Appl. Comput.*, 1998, pp. 318–326.

[17] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys, *The Traveling Salesman Problem*. Chichester, U.K.: Wiley, 1985.

[18] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*. Hoboken, NJ: Wiley, 1997.

[19] S. M. Thede, "An introduction to genetic algorithms," *J. Comput. Sci. Colleges*, vol. 20, no. 1, pp. 115–123, Oct. 2004.

[20] M. Kang and C. Han, "Solving the rural postman problem using a genetic algorithm with a graph transformation," in *Proc. ACM Symp. Appl. Comput.*, 1998, pp. 356–360.

[21] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 26, no. 1, pp. 29–41, Feb. 1996.

[22] M. Dorigo, G. Di Caro, and L. M. Gambardella, "Ant algorithms for discrete optimization," *Artificial Life*, vol. 5, pp. 137–172, 1999.

[23] K. M. Sim and W. H. Sun, "Ant colony optimization for routing and load-balancing: Survey and new directions," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 33, no. 5, pp. 560–572, Sep. 2003.

[24] WIKIPEDIA, Floyd-Marshall algorithm, 2006. [Online]. Available: http://en.wikipedia.org/wiki/Floyd-Marshall_algorithm

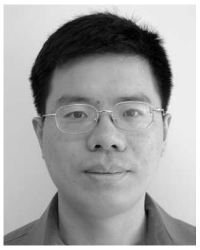
[25] R. Kulic and Z. Vukic, "Methodology of concept control synthesis to avoid unmoving and moving obstacles," *J. Intell. Robot. Syst.*, vol. 1, no. 37, pp. 21–41, May 2003.

[26] R. Kulic and Z. Vukic, "The control based on Lyapunov adaptation law to be improved by RBF neural network and behavioral cloning," in *Proc. Int. Conf. Computer as a Tool, EUROCON*, 2005, pp. 306–309.

Girma S. Tewolde (S'01) received the B.Sc. degree in electrical engineering from Addis Ababa University, Addis Ababa, Ethiopia, and the M.Eng.Sc. degree from the University of New South Wales, Sydney, Australia.

He is a Lecturer with Kettering University, Flint, MI, while working on his Ph.D. degree at Oakland University, Rochester, MI. His areas of interest are in sensor networks, embedded systems, mobile robotics, sensor and robot localization, task allocation, and evolutionary methods for optimization and engineering applications.

Mr. Tewolde is a graduate Student Member of the IEEE Computer Society and the IEEE Robotics and Automation Society.



Weihua Sheng (S'99–M'02) received the B.S. and M.S. degrees in electrical engineering from Zhejiang University, Hangzhou, China, in 1994 and 1997, respectively, and the Ph.D. degree in electrical and computer engineering from Michigan State University, East Lansing, in 2002.

He is currently an Assistant Professor with the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK. He is the holder of one U.S. patent. His research work has resulted in more than 60 papers in major journals and international conferences in robotics and automation. His current research interests include distributed robotic systems, mobile wireless sensor networks, process planning for advanced manufacturing technologies and embedded computing.

Dr. Sheng is a member of the IEEE Robotics and Automation Society.