Solving the Inspection Problem via Colored Traveling Salesman Tours

Stefan Edelkamp[1], Erion Plaku[2] , Christoph Greulich[1] and Mihai Pomarlan[1]

Abstract— **This paper describes an effective approach to solve the inspection problem via colored traveling salesman tours. Given an environment with obstacles and regions to be inspected, the approach plans a collision-free and dynamically-feasible motion trajectory that enables the vehicle to inspect all the regions of interest while reducing the distance traveled. A key aspect of the approach is the transformation of the inspection task into a colored traveling salesman problem (CTSP). This is achieved by generating a number of inspection points on the medial axis of each region in order to increase visibility and by grouping the inspection points according to their color. We also contribute a fast CTSP solver based on Monte-Carlo search to effectively find low-cost colored tours. Finally, a vehicle controller is employed in order to follow the planned tour. Experiments using different environments and inspection tasks provide promising validation.**

I. Introduction

As defects to objects such as pipeline leakage can result in tremendous economical loss, the inspection problem is one of the most important problems in modern robotics. Given a set of objects located in an environment with obstacles, the task is to find a least-cost obstacle-avoiding path that inspects all objects. Inspection is often expressed in terms of visibility, which may be limited by the sensor range, e.g., laser, camera.

A common approach to solve the inspection problem in polygonal environments is to first compute a minimal set of guards by solving the art gallery problem [1] and then to use an efficient TSP solver to compute a path for the robot towards visiting these guard positions. Both subproblems are NP-hard, so that the combined result might only be approximate for the entire task. Moreover, guards are often attached to walls and obstacles and thus difficult to reach.

Approaches to find guards often start with a collection of points that is not minimal but is easy to approximate and is expected to provide good coverage. The work in [2], [3] uses the skeleton of a 3D region, whose boundaries are defined by meshes, to define an initial set of guards which is then pruned using integer-linear programming solved through branch-and-bound. Another approach selects guards from skeleton points of a 2D environment, but prunes the guards less aggressively with a simple heuristic since their goal is to keep visibility on a moving target while minimizing the motion of the robot, rather than inspecting the entire workspace [4]. The work in [5] uses a decomposition of a 2D environment into convex polygons to quickly place the guards without attempting to minimize their number. Medial axis points are also considered good guard candidates, but medial axis itself is hard to compute exactly, so approximation algorithms based on Voronoi diagrams have been explored [6].

Visiting several polygonal regions through a shortest tour has been tackled in [7] through self-organizing map methods; a watchman route problem is handled by first providing a convex cover set of the environment. The work in [8] selects guards by creating a probabilistic roadmap [9] and then uses an approximate TSP solver on the shortest-path graph to compute a tour. Sampling-based path planning is also used in [10] to iteratively construct a set of guards for a 3D environment and create a cyclic path containing them, which is then smoothed and shortened using heuristics so as to not reduce the space visible from the guards. The work in [11] uses a sampling-based approach to integrate guard selection and pathfinding, so as to guarantee guards can be reached given the robot's dynamics. These sampling-based approaches are shown to be probabilistically complete and to asymptotically converge to the optimal path.

Contribution: Related work has only considered visibility and obstacle-avoidance constraints on the tour. In particular, the cited approaches do not provide a mechanism to specify constraints on the order in which the regions are inspected. Our work makes this possible by using colors to ensure that regions with the same color are inspected in order. This paper allows for the environment, obstacles, and regions of interest to be represented as a bitmap image. A crucial aspect of the proposed approach is the transformation of the inspection problem into a CTSP over an undirected, colored, and weighted graph. This is achieved by first generating waypoints that will enable the robotic vehicle to inspect each user-specified region of interest. The proposed approach adapts a grass-firing transformation algorithm in order to effectively generate these waypoints on the medial axis of each region and to ensure that the entire region can be inspected by visiting its waypoints. By grouping the waypoints according to their regions, the inspection problem is transformed into a CTSP where each point inherits the color of its region. A Monte-Carlo search is developed in order to efficiently solve the resulting CTSP and so compute a low-cost tour that enables the robotic vehicle to inspect all the regions. Finally, a controller is employed to drive the robotic vehicle from one waypoint to the next as specified by the tour.

The efficiency of the approach derives from its ability to generate a small number of waypoints while at the same time obtaining a good coverage of the regions that need

[1]Faculty of Mathematics and Computer Science, University of Bremen, Germany {edelkamp,greulich}@tzi.de, mpomarlan@yahoo.co.uk

[2]Department of Electrical Engineering & Computer Science, Catholic University of America, Wahington DC, USA plaku@cua.edu

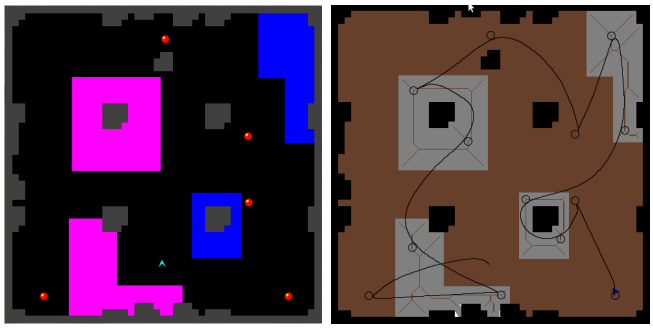

Fig. 1. (left) Input for the colored inspection problem. (right) Tour and collision-free path obtained by the proposed approach. The figure also shows the approximation of the medial axis generated by the grass-firing transformation algorithm as well as the inspection points generated by the approach. The white areas are not visible from the inspection points (in this example, the inspection quality was set to $\alpha = 0.99$). Figures are best viewed in color and on screen.

to be inspected. The proposed approach is evaluated using the PTSP benchmark set, which provides numerous problem instances, a vehicle simulator which takes into account the effects of the velocity, acceleration, and friction, as well as controllers to drive the vehicle. The experimental results provide promising validation, showing that the approach works well for a variety of environments and inspection tasks.

II. Problem Formulation

The environment is specified as a bitmap image $\mathcal{I}$ consisting of obstacles $O_1, \ldots, O_m$ and regions $R = \{R_1, \ldots, R_n\}$ that should be inspected, as shown in Fig. 1. Each region $R_i \in R$ defines a contiguous area in $\mathcal{I}$ which does not intersect any other region or obstacle. Obstacles are colored in `gray` and the unoccupied area $U = \mathcal{I} \setminus (R_1 \cup \ldots \cup R_n \cup O_1 \cup \ldots \cup O_m)$ is colored in `black`. The user specifies the color of each region $R_i \in R$, denoted by $\text{color}(R_i)$. The only requirement is that $\text{color}(R_i)$ is neither `gray` nor `black`. This provides a general definition as it allows for the same color to be used for different regions. The set of all region colors is defined as

$$\text{colors}(R) = \{\text{color}(R_i) : R_i \in R\}. \tag{1}$$

For a point $p \in \mathcal{I}$, $\text{color}(p)$ denotes its color.

Colors allow the user to group regions and specify desired constraints on how the robot should carry out the inspection. In particular, the robot is required to inspect all the regions in one color group[1] before inspecting a region from another color group.

The robot carries out the inspection by taking snapshots at different locations. The user can mark specific points $\hat{P} = \{\hat{p}_1, \ldots, \hat{p}_k\} \subset \mathcal{I} \setminus (O_1 \cup \ldots \cup O_m)$ that the robot should visit. The other inspection points should be automatically computed by the proposed approach. The robot can take a snapshot of a region $R_i \in R$ only when it is inside R_i. As a result, a point $q \in R_i$ is considered visible from $p \in \mathcal{I}$ iff $\overline{pq} \in R_i$, i.e., the straight-line segment from p to q remains entirely in R_i. Let $\text{vis}(R_i, p)$ denote the area of R_i that is visible from a point $p \in \mathcal{I}$, i.e.,

$$\text{vis}(R_i, p) = \{q : q \in R_i \wedge \overline{pq} \in R_i\}. \tag{2}$$

Note that $\text{vis}(R_i, p) = \emptyset$ when $p \notin R_i$. The area of R_i visible from a collection of points $p_1, \ldots, p_\ell \in \mathcal{I}$ is defined as

$$\text{vis}(R_i, p_1, \ldots, p_\ell) = \bigcup_{j=1}^{\ell} \text{vis}(R_i, p_j). \tag{3}$$

When considering all the regions in R, the visible area is defined as

$$\text{vis}(R, p_1, \ldots, p_\ell) = \bigcup_{R_i \in P} \text{vis}(R_i, p_1, \ldots, p_\ell). \tag{4}$$

The quality of a collection of points $p_1, \ldots, p_\ell \in \mathcal{I}$ is defined in terms of the fraction of the area in R that it enables the robot to inspect, i.e.,

$$\text{quality}(R, p_1, \ldots, p_\ell) = \frac{\text{area}(\text{vis}(R, p_1, \ldots, p_\ell))}{\text{area}(\cup_{R_i \in R} R_i)}. \tag{5}$$

A tour, denoted by $\text{tour}(p_1, \ldots, p_\ell)$, defines an ordering of the points $p_1, \ldots, p_\ell$. The notation Γ_i, where $\Gamma = \text{tour}(p_1, \ldots, p_\ell)$, refers to the i-th point in the tour. The inspection problem can now be stated as follows:

Definition 1: Given a desired inspection quality $0 < \alpha \leq 1$, a bitmap image $\mathcal{I}$ consisting of obstacles $O_1, \ldots, O_m$, regions $R = \{R_1, \ldots, R_n\}$, marked points $\hat{P} = \{\hat{p}_1, \ldots, \hat{p}_k\}$, and the start point $p_{\text{start}} \in \mathcal{I}$, compute

- a collection of points $p_1, \ldots, p_\ell \in \mathcal{I}$ where the robot is going to take the additional snapshots to carry out the inspection task,
- a tour $\Gamma = \text{tour}(p_{\text{start}}, p_1, \ldots, p_\ell, \hat{p}_1, \ldots, \hat{p}_k)$, which starts at p_{start} and specifies an ordering of the points $p_1, \ldots, p_\ell, \hat{p}_1, \ldots, \hat{p}_k$, and
- a path, denoted by $\text{path}(\Gamma)$, which connects the points in the order defined by Γ

such that

- $\text{quality}(R, p_{\text{start}}, p_1, \ldots, p_\ell, \hat{p}_1, \ldots, \hat{p}_k) \geq \alpha$,
- $\text{path}(\Gamma)$ is collision free, and
- color groups are inspected in order, i.e., $\forall i, j, k \in \{1, \ldots, |\Gamma|\} : \texttt{black} \notin \{\text{color}(\Gamma_i), \text{color}(\Gamma_j), \text{color}(\Gamma_k)\} \wedge i < j < k \wedge \text{color}(\Gamma_i) \neq \text{color}(\Gamma_j) \implies \text{color}(\Gamma_i) \neq \text{color}(\Gamma_k)$

The proposed approach seeks to reduce the number of points $p_1, \ldots, p_\ell$ it needs to carry out the inspection task. To further reduce the distance traveled by the robot, the points in $\text{tour}(p_{\text{start}}, p_1, \ldots, p_\ell, \hat{p}_1, \ldots, \hat{p}_k)$ will be connected with short collision-free paths.

III. Method

Pseudocode is shown in Alg. 1. The approach starts by generating several inspection points on the medial axis of each region $R_i \in R$ in order to increase the visible area (Alg. 1:1). As described in more detail in Section III-A, the approach uses a grass-firing transformation algorithm in order to efficiently approximate the medial axis.

A key aspect of the approach is the transformation of the inspection problem into CTSP over an undirected, colored,

[1] A color $c \in \text{colors}(R)$ defines the color group $\{R_i : R_i \in R \wedge \text{color}(R_i) = c\}$.

Algorithm 1 Pseudocode for the proposed approach

Input: α: inspection quality, $0 < \alpha \leq 1$; bitmap image $\mathcal{I}$ consisting of obstacles $O = \{O_1, \ldots, O_m\}$, regions $R = \{R_1, \ldots, R_n\}$ that should be inspected, points $\hat{P} = \{\hat{p}_1, \ldots, \hat{p}_k\}$ that should be visited; $p_{\text{start}} \in \mathcal{I}$: start point
Output: a short, collision-free, path that solves the inspection problem

1: inspectionPts $\leftarrow$ GENERATEINSPECTIONPOINTS$(\mathcal{I}, \alpha)$
2: $G = (V, E, \text{color}, \text{cost})$, where $V \leftarrow \text{inspectionPts} \cup \hat{P} \cup \{p_{\text{start}}\}$ and $E \leftarrow V \times V$
3: data $\leftarrow$ ALLPAIRSSHORTESTPATHS$(\mathcal{I}, V)$
4: **for** $(p_1, p_2) \in E$ **do** $\text{cost}(p_1, p_2) \leftarrow$ LENGTH(RETRIEVEPATH(data, p_1, p_2))
5: $\langle p_1, \ldots, p_r \rangle \leftarrow$ CTSPSOLVER(G, p_{start})
6: FOLLOWTOURVEHICLECONTROLLER$(p_1, \ldots, p_r)$

and weighted graph $G = (V, E, \text{color}, \text{cost})$. More specifically, the vertex set V consists of the inspection points generated by the approach, the start point p_{start}, and the user-defined points $\hat{P} = \{\hat{p}_1, \ldots, \hat{p}_k\}$ (Alg. 1:2). Each $p \in V$ inherits the color of the region that contains it, i.e., if $p \in R_i$ then $\text{color}(p) = \text{color}(R_i)$; if $p \in U$ then $\text{color}(p) = \texttt{black}$.

The set of edges contains all pairs, i.e., $E = V \times V$. The cost of an edge $(p_1, p_2) \in E$ is defined as the length of the shortest path from p_1 to p_2. An optimized implementation of Dijkstra's shortest-path algorithm [12] based on radix heaps [13] is used to compute the all-pairs shortest paths (Alg. 1:3–4).

A Monte-Carlo search is developed to solve the resulting CTSP (Alg. 1:5) by computing a low-cost tour Γ which starts at p_{start}, visits each vertex in V, and inspects the color groups in order. After computing Γ, the approach relies on a controller to move the robot from one inspection point to the next as specified by Γ (Alg. 1:6).

The rest of the section describes the main steps of the approach in more detail.

A. Generating the Inspection Points

Pseudocode for generating the inspection points is shown in Alg. 2. In order to increase the visible area, the approach seeks to generate the inspection points on the medial axis of each region $R_i \in R$. In a discrete grid, e.g., as imposed by the bitmap image $\mathcal{I}$, it is possible to approximate the medial axis by applying skeletal algorithms. One particularly efficient approach is the grass-firing transformation algorithm [14] which uses two passes over $\mathcal{I}$ to compute the Manhattan distances from the obstacles for each pixel (Alg. 2:1–5). Interpreting distances as brightness, the medial axis approximation is obtained by extracting the skeleton consisting of the locally brightest pixels (Alg. 2:6). Fig. 1 shows an example of the medial axis computed by the grass-firing transformation.

The points on the medial-axis skeleton are filtered in order to determine viable candidates for the inspection points

Algorithm 2 GENERATEINSPECTIONPTS$(\mathcal{I}, \alpha)$

Input: $\mathcal{I}$: bitmap image; α: desired inspection quality, $0 < \alpha \leq 1$
Output: a set of inspection points

1: $\mathcal{B} \leftarrow \text{zeros}(\text{height}(\mathcal{I}), \text{width}(\mathcal{I}))$ ◇ *grassfire transformation*
2: **for** $(i, j) \in \{0, \ldots, \text{height}(\mathcal{I}) - 1\} \times \{0, \ldots, \text{width}(\mathcal{I}) - 1\}$ **do**
3: **if** $\text{color}(\mathcal{I}(i, j)) \notin \{\texttt{black}, \texttt{gray}\}$ **then** $\mathcal{B}(i, j) \leftarrow 1 + \min\{\mathcal{B}(i-1, j), \mathcal{B}(i, j-1)\}$
4: **for** $(i, j) \in \{\text{height}(\mathcal{I}) - 1, \ldots, 0\} \times \{\text{width}(\mathcal{I}) - 1, \ldots, 0\}$ **do**
5: **if** $\text{color}(\mathcal{I}(i, j)) \notin \{\texttt{black}, \texttt{gray}\}$ **then** $\mathcal{B}(i, j) \leftarrow 1 + \min\{\mathcal{B}(i+1, j), \mathcal{B}(i, j+1)\}$
6: skeleton $\leftarrow$ extract pixels making up the most intense lines in the brightness map $\mathcal{B}$
7: skeleton $\leftarrow$ FILTER(skeleton) ◇ *select inspection points*
8: inspectionPts $\leftarrow$ skeleton; currScore $\leftarrow$ VISIBILITYSCORE$(\mathcal{I}, \text{inspectionPts})$
9: **for** $p \in$ skeleton **do**
10: newScore $\leftarrow$ VISIBILITYSCORE$(\mathcal{I}, \text{inspectionPts} \setminus \{p\})$
11: **if** newScore $\geq \alpha \vee$ currScore $=$ newScore **then**
12: inspectionPts $\leftarrow$ inspectionPts $\setminus \{p\}$; currScore $\leftarrow$ newScore
13: **return** inspectionPts

(Alg. 2:7). The filtering process removes points that are too close to the obstacles, imposes some minimum separation among inspection points, and gives preference to branching points, which have a degree of 3 or more in the skeleton graph. In our examples, the filtering process brought down the number of inspection points from thousands to 100–200. This is still a large number that would impose considerable runtime requirements on the CTSP solver and the vehicle controller.

To further reduce the number of inspection points while maintaining the desired inspection quality, candidates are also filtered according to the overall visibility score. The visibility score for a collection of points provides an efficient approximation of the inspection quality measure (Eqn. 5). The process we apply computes the visible area for each $p \in \text{inspectionPts}$ by casting light rays in all eight grid directions. To account for the color constraints, the light intensity becomes zero when encountering an obstacle or leaving the region $R_i \in R$ that contains p. The lightmaps computed for each $p \in \text{inspectionPts}$ are merged into an overall visibility map, where each $q \in \mathcal{I}$ is marked as visible when it is visible from at least one inspection point. VISIBILITYSCORE$(\mathcal{I}, \text{inspectionPts})$ (Alg. 2:8) is then computed by dividing the number of visible pixels by the number of pixels corresponding to regions in R.

The visibility score is used to further reduce the number of inspection points while maintaining the inspection quality (Alg. 2:9–12). For each candidate inspection point $p \in$

Algorithm 3 Computing a CTSP tour via Monte-Carlo Tree search

NRPA$(\text{level}, \text{iterations}, G = (V, E, \text{color}, \text{cost}), \text{global})$
1: best $\leftarrow (\infty, \emptyset)$
2: **if** level $= 0$ **then return** ROLLOUT(G, global)
3: backup[level] $\leftarrow$ global
4: **for** $i = 0, \ldots,$ iterations **do**
5: current $\leftarrow$ NRPA$(\text{level} - 1, \text{iterations}, G, \text{global})$
6: **if** current $<$ best **then** best $\leftarrow$ current; ADAPT$(\text{best}, \text{level}, \text{backup})$
7: global $\leftarrow$ backup[level]; **return** best

ROLLOUT$(G = (V, E, \text{color}, \text{cost}), \text{global})$
1: visited $\leftarrow (1, 0, \ldots, 0)$; tour[0] $\leftarrow 0$; tourSize $\leftarrow 1$
2: node $\leftarrow$ prev $\leftarrow c \leftarrow$ consecutive $\leftarrow 0$
3: **while** tourSize $< |V|$ **do**
4: prev $\leftarrow$ node; node $\leftarrow$ FITNESS$(\text{consecutive}, \text{color}, \text{global})$
5: tour[tourSize] $\leftarrow$ node; tourSize $\leftarrow$ tourSize $+ 1$
6: visited[node] $\leftarrow 1$; consecutive $\leftarrow$ color(node) $\neq$ black
7: **if** $|\{k : \text{visited}[k] = 1 \wedge \text{color}(k) = \text{color}(\text{node})\}| = \text{nrPts}(\text{color}(\text{node}))$ **then**
8: consecutive $\leftarrow 0$
9: $c \leftarrow c + \text{cost}(\text{prev}, \text{node})$
10: **return** (c, tour)

skeleton, we compare the visibility score with and without p. If the visibility score for $\text{inspectionPts} \setminus \{p\}$ is at least α, then p is removed from the inspection points since the inspection quality is still high. The point p is also removed when it does not impact the visibility score, i.e., it remains the same for $\text{inspectionPts} \setminus \{p\}$ and inspectionPts.

B. CTSP Solver

As described earlier in Section III, the inspection problem is transformed into a CTSP over an undirected, colored, and weighted graph $G = (V, E, \text{color}, \text{cost})$. Given that each vertex $p \in V$ inherits the color of the region that contains it, the objective is to compute a low-cost tour which starts at $p_{\text{start}} \in V$ and visits each vertex in V according to their color. Formally, a colored tour is defined as follows.

Definition 2: Let $G = (V, E, \text{color}, \text{cost})$ denote an undirected, colored, and weighted graph. Let $p_{\text{start}} \in V$ denote the start vertex. A sequence of vertices $\langle p_1, \ldots, p_r \rangle$ constitutes a valid colored tour if

- $\{p_1, \ldots, p_r\} = V$,
- $p_1 = p_{\text{start}}$,
- $\forall i \in \{1, \ldots, r-1\} : (p_i, p_{i+1}) \in E$, and
- $\forall i, j, k \in \{1, \ldots, r\} : \texttt{black} \notin \{\text{color}(p_i), \text{color}(p_j), \text{color}(p_k)\}$ and $i < j < k \wedge \text{color}(p_i) \neq \text{color}(p_j) \implies \text{color}(p_i) \neq \text{color}(p_k)$.

An optimal colored tour is a colored tour with minimum cost, where the cost of the tour is defined as the sum of the weights associated with the edges of the tour.

There are different approaches to solving TSPs depending on the size of the problem. Branch-and-bound techniques that rely on sophisticated lower bounds have proven effective in solving large unconstrained problems [15]. Integer programming branch-and-cut solvers have been shown to be most effective in finding optimal solutions for highly constrained TSPs [16]. Neighborhood local search algorithms are often used when the objective is to find satisfying tours [17].

Our CTSP solver is based on Monte-Carlo Tree search and is inspired by the Nested-Rollout-Policy-Adaptation (NRPA) algorithm [18]. Pseudocode is shown in Alg. 3. Essentially, there is the recursive driver NRPA, frequently calling a ROLLOUT procedure to construct a random tour that satisfies the coloring constraints. To generate this tour, the FITNESS procedure selects the successor according to a roulette wheel selection mechanism with respect to a policy which is maintained, copied, and updated at each level of the search. When better tours are found during the search, the level policy is adapted.

Note that the algorithm computes valid colored tours according to Definition 2, and that no valid tour is rejected a priory. If N is the number of inspection points, the Rollout algorithm has a running time of $O(N)$. For fixed number of iterations, NRPA applies $\text{iterations}^{\text{level}}$ rollouts.

C. Following the CTSP Tour

Once a tour Γ is computed, a vehicle controller is employed to drive the robot from one inspection points to the next in the order defined by Γ. The vehicle is controlled by setting the acceleration (on or off) and turning the vehicle left, right, or keeping it straight. The state of the vehicle at time step t is described by the orientation d_t, velocity v_t, and position p_t. The equations of motions are defined as in [19], i.e.,

$$d_{t+1} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} d_t, v_{t+1} = (v_t + d_{t+1} a_t K) L,$$
$$p_{t+1} = p_t + v_{t+1},$$

where θ is a fixed angle which determines the rotation at each time step (θ is 0 if the action is to go straight), K is the acceleration constant, a_t is 1 if the acceleration is on and 0 if it is off, and L is the friction loss factor. The vehicle inertia is preserved, which makes the navigation more challenging. This paper leverages the vehicle controller in the PTSP setting [19] which takes into account the vehicle dynamics. The execution model of the controller is real-time, in which actions have to be committed a rate of about 40ms. The controller has numerous applications ranging from modeling non-player characters in interactive games to approximating steering behavior of mobile robots. The testbed has raised significant interest in combined task and motion planning and has the potential to cross-fertilize the research within AI search and planning.

IV. EXPERIMENTS AND RESULTS

The approach is evaluated using several maps from the PTSP benchmark set [19], as shown in Fig. 2. Experiments are conducted under two scenarios. In the first scenario, the robot

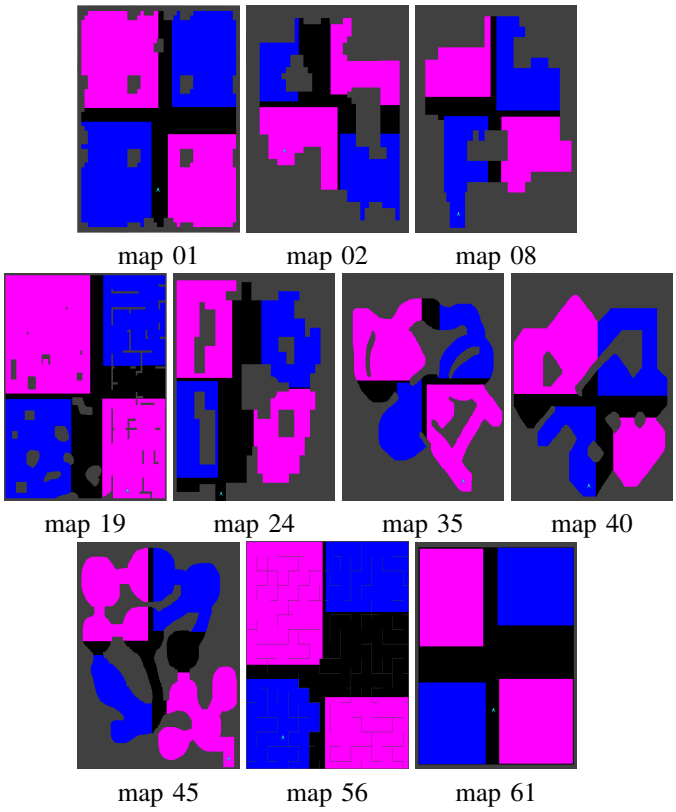

Fig. 2. Maps from the PTSP benchmark set adapted for the inspection problem with colored regions. Figures best viewed in color and on screen.

Map	Runtime	IPs	Cost	Steps
01	0.95s ± 0.24s	4	767.5 ± 0	583.8 ± 11.5
02	0.45s ± 0.02s	6	1171.5 ± 0	967.5 ± 84.9
08	0.32s ± 0.01s	4	1965.4 ± 0	2142.3 ± 76.3
19	1.69s ± 0.10s	13	2451.6 ± 57.8	2144.5 ± 76.0
24	0.91s ± 0.02s	12	1818.4 ± 0	1577.3 ± 27.7
35	0.57s ± 0.04s	11	1478.4 ± 0	1171 ± 50.5
40	0.38s ± 0.03s	5	975.8 ± 0	804 ± 16.0
45	0.68s ± 0.01s	12	1792.1 ± 0	1506.3 ± 24.5
56	6.14s ± 0.50s	63	7606.5 ± 321.6	11848.3 ± 561.8
61	0.72s ± 0.14s	1	123.3 ± 0	162 ± 6

Map	Runtime	IPs	Cost	Steps
01	1.98s ± 0.01s	9	1727.5 ± 0	1387.7 ± 44.4
02	2.69s ± 0.12s	12	1930.2 ± 0	1543.8 ± 36.6
08	1.98s ± 0.05s	10	1614.0 ± 0	1422 ± 116.6
19	11.74s ± 0.45s	30	3558.0 ± 55.5	4324.1 ± 186.8
24	4.68s ± 0.24s	17	2546.9 ± 0	2187.4 ± 76.9
35	2.36s ± 0.08s	12	1863.4 ± 10.3	1607 ± 90.8
40	1.97s ± 0.02s	9	1566.5 ± 0	1380.6 ± 62.3
45	2.96s ± 0.03s	14	2348.9 ± 7.9	2323.2 ± 161.8
56	34.3s ± 0.55s	51	8352.8 ± 352.6	(stuck)
61	1.53s ± 0.08s	4	1323.6 ± 0	1082.3 ± 19.4

TABLE I

RESULTS WHEN SOLVING (TOP) THE STANDARD INSPECTION PROBLEM (BOTTOM) THE INSPECTION PROBLEM WITH COLORED REGIONS. COLUMNS INDICATE THE FOLLOWING: (MAP) MAP NUMBER; (RUNTIME) TIME TO PROCESS THE INPUT IMAGE, GENERATE THE INSPECTION POINTS, AND COMPUTE THE INSPECTION TOUR; (IPS) NUMBER OF INSPECTION POINTS GENERATED BY THE APPROACH; (COST) ESTIMATED TRAVELED DISTANCE WHEN FOLLOWING THE INSPECTION TOUR; (STEPS) THE NUMBER OF STEPS TAKEN BY THE CONTROLLER. RESULTS ARE GIVEN AVERAGED OVER 10 RUNS, ± STANDARD DEVIATION. WAYPOINT COUNTS ARE THE SAME FOR ALL RUNS OF A MAP-PROBLEM PAIR, HENCE NO STANDARD DEVIATION.

is required to inspect the area not occupied by obstacles, i.e., $\mathcal{I} \setminus (O_1 \cup \ldots \cup O_m)$. This corresponds to the standard inspection problem where there is only one color. In the second scenario, the robot is required to inspect only the colored regions. Each region was assigned one of two colors, and regions on adjacent corners have different colors, to force the solver to trade-off distance traveled versus enforcing the color sequence constraint. In all the experiments, the inspection quality α was set to 0.99. Every experiment is run 10 times; results report the average and standard deviation statistics.

Table I shows the results of the two sets of experiments: the standard inspection problem and the inspection problem with colored regions. The results show the approach works well under different scenarios and environments, generating low-cost tours. The runtime, which includes the time to process the input image, generate the waypoints, and compute the inspection tour (Alg. 1:1–5), is about $1s$ for the standard inspection problems and generally between $1.5s$ and $3s$ for the inspection problems with colored regions. There are also quite challenging environments, such as the one provided by Map 56, which are characterized by numerous narrow passages. In these cases, considerably more inspection points are needed, which increases the runtime of the CTSP solver. The narrow passages also make it difficult for the controller to follow the inspection tour. All the experiments were run on a machine with an Intel®Core™i5-3210M CPU 2.50GHz $\times$ 4 with 4GB RAM. The code is written in Java.

We have also investigated how the iterations parameter in the CTSP solver (Alg. 3) affects convergence to a solution. In particular, we looked at whether it is advantageous to have many iterations at the topmost level of the NRPA with small iteration counts at lower levels. For levels below the top, we varied iterations between 2 and 15. For the top-level we set the iteration count to 3000 divided by iterations^2. We have tried several maps, and run each map 10 times for each parameter setting to get some statistics on performance. For all runs, the level parameter was kept at 5.

Fig. 3 shows the score curves we obtained for the map19 problem with all freespace designated as an inspection area. The shapes of the score curves are similar for other maps. The x-axis is the number of rollouts done from the lowest level, and is an indicator of computation time (for the same map, rollouts take the same time regardless of iterations because they are below the search levels at which the search iterates). The y-axis is the score of the tour, and is what the solver optimizes for. Note that this is the score determined by the solver, not computed by the controller when actually taking the tour. The figure shows averages over the ten runs for different parameter settings.

The score curves tend to show a region of quick improvement, followed by a lower plateau where the search may get trapped in local minima; solvers with lower values of

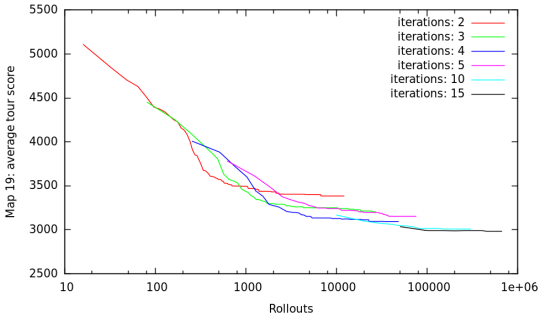

Fig. 3. Influence of the iterations parameter for levels below the topmost one over search convergence; plot of tour score vs. number of rollouts. Curves are averaged over ten runs with the same parameter setting on the same map. Results are shown for map19 (Fig. 2).

iterations are particularly prone to this, and tend to have more variable performance from run to run. However, in the region of fast improvement they can obtain better scores with fewer rollouts than solvers with higher values of the iteration parameter. We suspect this is because improvements are "low-hanging fruit," easy to find at first, and benefit more from trying out a few different solution approaches (many top level iterations) rather than trying to refine already found ones (many bottom level iterations).

V. Discussion

This paper developed an effective approach for the inspection problem with colored regions. A key aspect of the approach was the transformation of the inspection problem into a CTSP over an undirected, colored, and weighted graph. The approach relied on a grass-firing transformation algorithm to effectively approximate the medial axis of each colored region. The approach relied on ray casting and visibility scores to generate inspection points on the medial axis that would enable the robot to achieve the desired inspection quality. An effective CTSP solver was developed based on Monte-Carlo tree search in order to compute low-cost inspection tours. Experiments using different environments and inspection tasks provided promising validation.

This work opens up several venues for future research. One direction is to integrate the approach with sampling-based motion planning so that it can work for complex robotic systems with non-linear dynamics. The colored tour would serve to guide the sampling-based motion planner, which in turn would provide information about the feasibility of each tour. Another direction is to extend the approach to 3D environments where the inspection is carried out by an aerial vehicle. We will also seek to extend the approach to multiple robots working together.

References

[1] D. Borrmann, P. J. De Rezende, C. C. De Souza, S. P. Fekete, S. Friedrichs, A. Kröller, A. Nüchter, C. Schmidt, and D. C. Tozoni, "Point guards and point clouds: Solving general art gallery problems," in *ACM Symposuim on Computational Geometry*, 2013, pp. 347–348.

[2] W. Yu, M. Li, and X. Li, "Optimizing pyramid visibility coverage for autonomous robots in 3d environment," *Control and Intelligent System*, vol. 42, pp. 9–15, 2014.

[3] X. Li, W. Yu, X. Lin, and S. S. Iyengar, "On optimizing autonomous pipeline inspection in 3D environment," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 223–233, 2012.

[4] I. Ardiyanto and M. J. Toyohashi, "Visibility-based viewpoint planning for guard robot using skeletonization and geodesic motion model," in *IEEE International Conference on Robotics and Automation*, 2013, pp. 660–666.

[5] G. D. Kazazakis and A. A. Argyros, "Fast positioning of limited-visibility guards for the inspection of 2d workspaces," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002, pp. 2843–2848.

[6] R. Fabbri, L. F. Estrozi, L. Da, and F. Costa, "On voronoi diagrams and medial axes," *Journal of Mathematical Imaging and Vision*, vol. 17, pp. 27–40, 2002.

[7] J. Faigl, V. Vonásek, and L. Preucil, "A multi-goal path planning for goal regions in the polygonal domain," in *European Conference on Mobile Robots*, 2011, pp. 171–176.

[8] T. Danner and L. E. Kavraki, "Randomized planning for short inspection paths," in *IEEE International Conference on Robotics and Automation*, 2000, pp. 971–976.

[9] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[10] B. Englot and F. Hover, "Sampling-based coverage path planning for inspection of complex structures," in *International Conference on Automated Planning and Scheduling*, 2012, pp. 29–37.

[11] G. Papadopoulos, H. Kurniawatia, and N. M. Patrikalakis, "Asymptotically optimal inspection planning using systems with differential constraints," in *IEEE International Conference on Robotics and Automation*, 2013, pp. 4126–4133.

[12] E. W. Dijkstra, "A note on two problems in connexion with graphs." *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[13] R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan, "Faster algorithms for the shortest path problem," *Journal of the ACM*, vol. 37, no. 2, pp. 213–223, 1990.

[14] H. Blum, "A transformation for extracting new descriptors of shape," in *Models for the Perception of Speech and Visual Form*, 1967, pp. 362–380.

[15] W. Zhang, "Truncated branch-and-bound: A case study on the asymmetric TSP," in *AAAI Spring Symposium on AI and NP-Hard Problems*, 1993, pp. 160–166.

[16] N. Ascheuer, M. Fischetti, and M. Grötschel, "Solving the asymmetric travelling salesman problem with time windows by branch-and-cut," *Mathematical Programming*, vol. 90, no. 3, pp. 475–506, 2001. [Online]. Available: http://dx.doi.org/10.1007/PL00011432

[17] R. Bent and P. V. Hentenryck, "A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows," *Computers & Operations Research*, vol. 33, no. 4, pp. 875–893, 2006.

[18] C. D. Rosin, "Nested rollout policy adaptation for monte carlo tree search," in *International Joint Conference on Artificial Intelligence*, 2011, pp. 649–654.

[19] D. Perez, P. Rohlfshagen, and S. M. Lucas, "The physical travelling salesman problem: WCCI 2012 competition," in *IEEE Congress on Evolutionary Computation*, 2012, pp. 1–8.

[20] M. Phillips, A. Dornbush, S. Chitta, and M. Likhachev, "Anytime incremental planning with e-graphs," in *IEEE International Conference on Robotics and Automation*, 2013, pp. 2444–2451.

[21] R. Wein, J. P. Van Den Berg, and D. Halperin, "The visibility-Voronoi complex and its applications," in *ACM Symposium on Computational Geometry*, 2005, pp. 63–72.

[22] F. Aurenhammer and R. Klein, "Voronoi diagrams," in *Handbook of Computational Geometry*. Elsevier Science, 2000, pp. 201–290.

[23] S. Ghosh and D. M. Mount, "An output-sensitive algorithm for computing visibility graphs," *SIAM Journal on Computing*, vol. 20, no. 5, pp. 888–910, 1991.