A GRASP Metaheuristic for the Coverage of Grid Environments with Limited-Footprint Tools

Alessandro Riva
Politecnico di Milano
Piazza Leonardo da Vinci, 32
Milano, Italy
alessandro.riva@polimi.it

Francesco Amigoni
Politecnico di Milano
Piazza Leonardo da Vinci, 32
Milano, Italy
francesco.amigoni@polimi.it

ABSTRACT

Coverage of known environments is a task involved in several applications of autonomous mobile robots, like patrolling, search and rescue, and cleaning. The (single robot) coverage problem can be formulated as that of finding the optimal tour that, when followed, allows a robot to cover with its tool (e.g., a sensor or a brush) all the points of the free space of a given environment. Most of the current methods for coverage discretize the environment in cells, possibly of different shapes. In this paper, we consider a setting in which the environment is represented as a grid of equal square cells and in which a robot has a tool with limited range and angular field of view, able to cover a set of cells from a given pose. We propose an efficient covering method based on a on Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic approach that iteratively constructs a feasible solution and tries to improve it through local search. Results of experimental activities show that the proposed method produces solutions of better quality than those of a state-of-the-art method, in an efficient way.

Keywords

Robotic coverage, Greedy randomized adaptive search procedure, Grid environments

1. INTRODUCTION

In the field of autonomous mobile robotics, the problem of coverage is encountered in many applications, like search and rescue [4], patrolling [6], target detection [17], and ploughing [16]. A general formulation of the *coverage problem* in a given planar environment involves a robot (or multiple robots) equipped with a covering tool, whose footprint overlays a finite area, that has to sweep the entire area of the environment, optimizing an objective function, usually related to distance travelled or time taken [8]. Examples of covering tools include the brush of a vacuum cleaner, the blade of a lawnmower, and the laser range scanner of an exploring robot. Mainstream approaches discretize the environment in some way, for example decomposing it in cells with different shapes (e.g., according to the presence of obstacles and to the covering tool) or with regular shapes [15]. The approaches that consider regular discretizations of the environment in *grids* of squared cells are suitable for modelling long-range perception tools like laser range scanners and remote gas sensors. In this case, the tool can cover several cells when the robot is at a given pose. Although these settings are typically considered in autonomous mobile robotics, the corresponding coverage problem is not much studied, the main results being in [3, 21]. In both formulations, with irregular and regular cell decompositions, the coverage problem is NP-hard for general environments [7, 10]. In the attempt to find (suboptimal but) good solutions, most methods [3, 9, 11, 12] independently address two subproblems: (a) finding a set of poses that, when visited, allow the tool to cover all the environment and (b) finding the optimal tour through these poses. Clearly, the optimization of the two subproblems is not guaranteed to produce a high-quality solution for the coverage method.

In this paper we propose a novel method for an autonomous mobile robot that has to cover a grid environment, which exploits the metaheuristic approach based on Greedy Randomized Adaptive Search Procedure (GRASP) [19] to address the two above subproblems at the same time. GRASP methods have been successfully used to solve related problems, like the Travelling Salesman Problem (TSP) and the orienteering problem. Here, we originally apply it to the coverage problem in the attempt to overcome the two-subproblem optimization performed by most of the current methods. More precisely, we assume to know a planar environment represented as a grid and to have a robot equipped with a covering tool characterized by a limited range r and a limited angular field of view θ. Our goal is to find a closed sequence of poses (a tour) in the grid such that the union of the areas covered at these poses completely covers the environment and that the tour is the fastest one. In our model, the time required to complete a tour is composed of the time required for moving and of the time required for sensing. This second quantity accounts for situations in which the acquisition of data is not almost-instantaneous (as in the case of laser range scanners) but requires some time. For instance, a remote gas sensor like that used in [3] can take dozens of seconds to sweep an area bounded by $r = 5$ meters and $\theta = 45$ degrees. In taking into account the sensing time, our approach differs from many methods proposed in the literature (e.g., [21]).

Our approach works in two phases. First, an initial covering tour is calculated, using a random greedy procedure. Then, this tour is improved by local search. These two phases are iterated until a termination criterion is satisfied. Experimental results show that our proposed GRASP

Appears in: *Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*

method consistently outperforms the method in [3], suggesting that tackling together subproblems (a) and (b) can be more efficient.

2. RELATED WORK

The problem of finding the optimal path for covering the area of a planar environment with an autonomous mobile robot equipped with a covering tool has been addressed by several approaches that can be classified in different ways [8,15]. A first broad classification concerns the nature of the problem state space, namely the model of the environment, which can be continuous or discrete.

The continuous problem formulation, usually called Watchman Route Problem (WRP), has been extensively studied from a geometrical point of view, modeling the environment as a polygon (with holes). In the general case, the WRP is a NP-hard problem (see [10] for the proof), and cannot be approximated in polynomial time within a factor of $c \log n$, where n is the number of vertices of the environment polygon and $c > 0$, unless P=NP [18]. Efforts to tackle the WRP usually rely on strong assumptions, which are often far from practical robot implementations, especially when the covering tool is a sensor. They include time-continuous perception and unlimited range of the covering tool. Sometimes, also a perception angle of 360 degrees is assumed, considering that the robot perceives the surrounding environment in an omnidirectional fashion, which is not the case for most cameras, laser range scanners, and remote gas sensors (unless multiple perceptions are merged).

In the discrete formulation of the coverage problem, the environment is modeled as a graph or a grid. In these settings, the range of the covering tool of the robot can either just cover a single vertex (grid cell) or cover sets of graph vertices (grid cells) with a single perception. Both problems are NP-hard, since they generalize the TSP on grids. As a consequence, suboptimal approaches have been developed. In the literature, the former sensing model has been largely adopted and studied over the past decades (e.g., in [1, 14, 23]), while, only recently the latter model has been considered and analyzed to solve problems such as camera surveillance [21] and gas detection [3].

In [21], the authors consider a grid environment on which they build an observation graph, enriching each free cell with a discrete set of possible orientations of the robot. On this graph, a Mixed Integer-Linear Program (MILP) is formulated and solved. The covering tool the authors consider is a camera, even though their approach works for any arbitrary covering function (i.e., for any way to decide whether a robot from a given pose covers a grid cell or not). To tackle large problem instances, the authors propose to cluster groups of free cells. Their approach is experimentally assessed on strongly-structured rectilinear environments and seems to scale well until instances of medium sizes (the paper reports that the algorithm takes 382, 833 seconds to find a solution on a grid with 314 cells).

In [3], the problem setting is the same of [21] (and the same of this paper). The coverage problem is decomposed in two subproblems, as discussed in the previous section. In this case: (a) finding a minimum set of vertices (on the observation graph) from which the perceptions cover the environment and (b) finding of the shortest tour among those locations (i.e., a TSP). In order to tackle large problem instances, the authors propose a convex relaxation "cSPP",

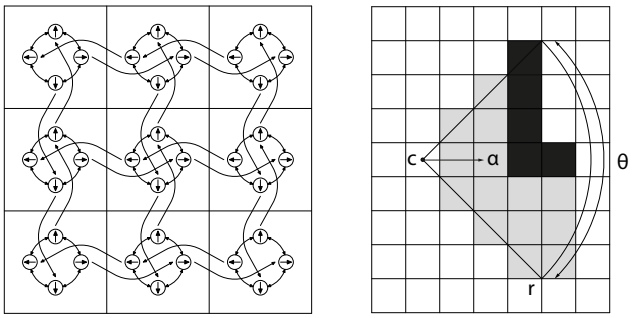

Figure 1: An example of movement graph on a 9×9 grid, with $\alpha \in \{0, \frac{\pi}{2}, \pi, \frac{3}{2}\pi\}$ (left). An example of covering function with $r = 5$ and $\theta = \frac{\pi}{2}$. In this case, the covering tool does not cover free cells behind obstacle cells (right). (Figure drawn by authors, after [3].)

which is able to quickly find solutions to (a). The proposed approach is shown to work in realistic-size environments and will constitute a benchmark for our method.

A well-studied variant of the coverage problem involves multiple robots. For example, in [20], a multirobot version of the method presented in [21] is proposed.

In [22], an adversarial formulation of the coverage problem for grid environments is presented. The robot covers the cell in which it is currently located and each grid cell has a known probability to block (or destroy) the robot. Instead of minimizing the traveled distance, the objective is to maximize the probability to survive, while covering the environment.

Against the state-of-the-art depicted above, in this paper we originally propose to use a GRASP method [19] for the single robot coverage problem in grid environments with an arbitrary covering function between robot poses and grid cells.

3. PROBLEM STATEMENT

We consider a finite planar environment discretized as a grid of identical squared cells, which can be either free cells F or cells occupied by obstacles O that are not traversable by the robot.

Given a grid, the robot state space is discretized in a set of poses V. Each pose $v \in V$ is a pair (c, α), where $c \in F$ is a free cell (we assume that the robot is always located at the center of the cell) and α is the orientation angle of the robot (according to some global reference frame) taken from a finite set of values. For sake of simplicity, let us assume that $\alpha \in \{0, \frac{\pi}{2}, \pi, \frac{3}{2}\pi\}$, but the approach can be generalized for any angle discretization.

The robot state space is then conveniently described by a graph, as shown in Figure 1. Formally, the *movement graph* $G = (V, E)$ is a directed weighted graph, where V is a set of vertices (coinciding with the set of poses) and E is a set of edges that represent the movements the robot can perform. More precisely, let $\alpha^{(1)}, ..., \alpha^{(h)}$ be the discretized values of the robot orientation angle, with $\alpha^{(1)} < ... < \alpha^{(h)}$. Then E contains a *rotation edge* from a vertex $v = (c, \alpha^{(i)})$ to a vertex $v' = (c, \alpha^{(i+1)})$ and a dual one from v' to v. Moreover, E contains a *translation edge* from a vertex $v = (c, \alpha)$ to a vertex $v' = (c', \alpha)$ if c and c' are neighbors on the 4-connected grid and α points along the direction from the

Algorithm 1 GRASP(L)

1: **while** stopping criterion not satisfied **do**
2: $S_{init} \leftarrow$ GreedyRandomizedSolution(L)
3: $S_{local} \leftarrow$ LocalSearch(S_{init})
4: $S_{incumbent} \leftarrow$ UpdateIncumbent(S_{local})
5: **return** $S_{incumbent}$

center of c to the center of c'. The weight associated to each edge is the time needed by the robot to rotate or move straight, which depends on the locomotion capabilities of the robot.

We define a *covering function* $C : V \rightarrow 2^F$ that returns the set of covered cells $C(v)$ from a given pose $v \in V$. Even though our approach can be applied to arbitrary covering functions, for simplicity in the following we consider a sensing tool with limited range r and limited angle θ. Hence, the covering function C is defined as follows: the robot in a pose $v = (c, \alpha)$ covers a cell c' if the line segment connecting the centers of c and c' is in the circular sector of radius r and angle θ, centered in c and around α, and does not cross any occupied cell. Figure 1 shows an example of this covering function. A perception performed from a pose v thus covers the cells in $C(v)$ and we assume it takes $T(v)$ time, which depends on the sensing capabilities of the robot.

A tour is a closed path on the movement graph G (the robot is required to come back to the starting point) and is represented as an ordered sequence of vertices (or poses) $S = [v_1, ..., v_k]$, such that $v_i \in V$, $i = 1, ..., k$. The idea is that the robot makes a perception from each v_i.

The cost of a tour S is the sum of the traveling times and the sensing times:

$$\sum_{i=1}^{k} d(v_i, v_{i+1}) + \sum_{i=1}^{k} T(v_i) \quad (1)$$

where $d(v, v')$ is the least cost for moving from v to v' on G and v_{k+1} corresponds to v_1. A *solution* to the coverage problem is a tour $S = [v_1, ..., v_k]$, such that $\cup_{i=1}^{k} C(v_i) = F$. The objective is to find the optimal solution, namely the solution with minimum cost.

4. A GRASP METHOD

In this section, we define our GRASP method for the coverage problem formulated above. A GRASP is a simple and effective metaheuristic approach [19] designed to find good solutions to NP-hard problems in a limited amount of time, that can be tuned through a stopping criterion, as shown in Algorithm 1.

A GRASP iteration starts with creating a greedy randomized solution from scratch in the following way. The function GreedyRandomizedSolution($\cdot$) iteratively computes the cost of each local choice, according to a greedy and problem-dependent criterion. At each iteration the solution is built by selecting one of the best L choices at random (where L is a parameter). The solution so created is then improved by a local search, which applies a set of possible local moves to reach a local optimum, following a best-improvement policy. Along this pipeline, a GRASP produces a (possibly different) solution to the problem at each iteration, which can be discarded or become the incumbent solution, that is, the current best solution found over all the previous iterations.

Algorithm 2 GreedyRandomizedSolution(L)

1: $S_{init} = [\,]$
2: **while** S_{init} does not cover F **do**
3: $Q \leftarrow$ best L vertices $\notin S_{init}$ w.r.t. Criterion (2)
4: $v \leftarrow$ pick at random one vertex from Q
5: $S_{init} \leftarrow$ push back v
6: **return** S_{init}

Algorithm 3 LocalSearch(S_{init})

1: $S'_{init} \leftarrow$ 2-OPT(S_{init})
2: $S_{l1} \leftarrow$ ExchangeProcedure(S'_{init})
3: $S_{l2} \leftarrow$ EliminationProcedure(S_{l1})
4: $S'_{l2} \leftarrow$ 2-OPT(S_{l2})
5: **return** S'_{l2}

It is worth to point out that each iteration is largely independent from the others and the algorithm can be easily parallelized, achieving a linear speed up, as discussed in [13].

4.1 Randomized Greedy Solution

To construct an initial randomized greedy solution S_{init} for our problem, we first define a greedy strategy for coverage. Greedy strategies have been largely studied and evaluated in the field of on-line coverage (i.e., exploration and mapping [2]), where the state-of-the-art methods use greedy criteria based on combinations of distance and expected covered area, in order to select the next pose to visit in a partially explored environment. Taking inspiration from these works, we define a greedy strategy for our problem that, at each iteration, creates an *elite set* from which the next vertex $v \in V$ is chosen. More precisely, given a (partial) tour $S_{init} = [v_1, ..., v_i]$ and the set of already-covered vertices $C_i = \cup_{j=1}^{i} C(v_j)$, the next vertex v_{i+1} added to S_{init} is selected randomly from the best L vertices, ordered according to:

$$\underset{v \notin S_{init}}{\arg\max} \ \frac{|C(v) \setminus C_i|}{d(v_i, v)} \quad (2)$$

where $d(v_i, v)$ is the distance between the vertex v and the last vertex v_i of S_{init} on the movement graph G and $|C(v) \setminus C_i|$ is the number of newly covered vertices if the robot moves to v (note that, differently from on-line problems, we know the exact amount of new covered vertices). In the selection of the first vertex of S_{init}, the distance $d(\cdot, \cdot)$ is neglected and put equal to 1. As shown in Algorithm 2, at each iteration, a vertex $v_{i+1} = v$ is chosen randomly from the L best vertices stored in an elite set Q, obtaining a new partial solution $S_{init} = [v_1, ..., v_{i+1}]$.

The GreedyRandomizedSolution($\cdot$) procedure stops when $S_{init} = [v_1, ..., v_k]$ covers all the free cells, namely when $\cup_{i=1}^{k} C(v_i) = F$. When this happens, S_{init} is returned.

4.2 Local Search

The local search procedure consists of an exchange phase and a removal phase. Both work on feasible solutions and, at each step, they check whether a certain move can be applied or not.

The local search first runs the *exchange phase*, which tries to substitute a vertex in S_{init} with a vertex in V but not in the tour S_{init}. Among all the possible substitutions, the

one that produces the maximum decrease of the tour cost is selected. Once the solution cannot be further improved by this move (that is, a local optimum is reached) the solution obtained S_{l1} is returned and the second phase is run.

In the second phase, called *removal phase*, the algorithm tries to remove vertices from S_{l1}. If, excluding a vertex v_i from S_{l1}, the solution is still feasible (i.e., it covers all the free cells in F), v_i is removed from the tour and its preceding and following vertices are directly linked, obtaining $S_{l2} = [v_1, ..., v_{i-1}, v_{i+1}, ..., v_k]$. Also in this case, among all the possible removals, the one that maximally decreases the cost of the solution is selected, until a local optimum is reached. The idea behind this second phase is that, especially in the initial steps of the greedy solution, some unnecessary vertices could be selected. This redundancy provides a certain degree of freedom in the exchange phase, which, however, cannot modify the number of vertices in the tour. For this reason, once the exchange phase finds a local optimum, we try to simplify as much as possible this tour, removing unnecessary vertices to improve the tour cost.

Before and after running the local search, a simple 2-OPT procedure [5] is applied to S_{init} and S_{l2}, respectively. This local optimization, originally designed for the TSP, reverts some parts of the tour, aiming at improving the overall cost. More precisely, for each pair of vertices in the tour, the 2-OPT procedure reverts the order of the vertices, i.e., given $v_i, v_j \in V$ with $i < j$, the portion of the tour $[v_i, v_{i+1}, ..., v_{j-1}, v_j]$ becomes $[v_j, v_{j-1}, ..., v_{i+1}, v_i]$. If the overall cost is reduced, the changing is accepted, discarded otherwise. The reason behind this further local optimization is twofold: not only the final solution cost is lowered in many cases, but improving the solution S_{init} also reduces the computational cost of the exchange phase, which, in general, may take a while to reach a local optimum (as we will see in Section 4.3).

The local search procedure is summarized in Algorithm 3.

4.3 Convergence and Complexity

We now analyze the complexity of the GRASP method defined above, from a worst-case point of view, with respect to the total number of vertices in the movement graph, $n = |V|$ and to the size k of the tour initially found by GreedyRandomizedSolution(·). Such a complexity analysis has to take into account the convergence of the local search to a local optimum.

First, note that the creation of the initial greedy solution and all the local search procedures are in sequence. This means that the complexity of the whole GRASP is the sum of each single procedure complexity, times the number of iterations (which we assume to be a constant number). Hence, the asymptotical GRASP complexity is the complexity of the procedure with the highest computational cost.

Let us start analyzing GreedyRandomizedSolution(·). The complexity of a single iteration is the complexity of evaluating all the vertices not yet in the tour, plus the complexity of picking at random a vertex from the best L ones. When evaluating the function in (2), the distance can be computed in $O(n^2)$ by using Dijkstra's algorithm, while the number of newly covered vertices can be obtained in $O(n^2)$ by incrementally storing those already covered. The set of L-best vertices can be created in $O(n)$ and random picking a vertex and appending it to the partial tour are both constant-time operations. Since the number of iterations is, by definition, equal to k, the whole procedure complexity is $O(kn^2)$.

The 2-OPT(·) procedure can be computed in $O(k^3)$ for a straightforward implementation.

Let us leave aside the exchange phase for a moment and focus on the removal phase. For each vertex v_i in the tour, the EliminationProcedure(·) computes the improvement in removing v_i, by simply subtracting the new local cost of $d(v_{i-1}, v_{i+1})$, to the old one, $d(v_{i-1}, v_i) + d(v_i, v_{i+1})$ (taking into account also the sensing cost of v_i). Each elimination move must be feasible (that is, the new tour has to still cover all the free cells). A feasibility check can be done by checking whether all the cells covered by v_i (whose number is a constant depending on r and θ) are covered by at least one other vertex in the tour. If the covering function is implemented through a sparse matrix, the feasibility check can be done in $O(k)$ (note that no other procedure modifies the tour size before the removal phase). The removal of a vertex that produces the largest improvement without harming tour feasibility can be computed in $O(k^2)$ for a single elimination move. Since the maximum number of vertices that can be removed is bounded by k, the removal phase converges to a local optimum in $O(k)$. Thus, the EliminationProcedure(·) complexity is $O(k^3)$.

We now focus on the complexity of a single move in the exchange phase. ExchangeProcedure(·) tries to exchange a vertex not in the tour with a vertex in the tour. All the possible combinations are $O(kn)$ and a feasibility check can be done in $O(k)$, following a reasoning similar to the one of the removal phase. Also in this case, the maximum improvement exchange can be identified during the computation of the values for all possible exchanges, giving a single move complexity of $O(k^2 n)$. Let us now bound the convergence complexity. Let γ be the cost of the input solution for ExchangeProcedure(·), γ_k^* be the minimum cost for a tour of size k, δ be the smallest sensing time $\min_{v \in V} T(v)$ plus the smallest edge weight in the movement graph G, and Δ be the largest sensing time $\max_{v \in V} T(v)$ plus the largest edge weight in the movement graph G. It is easy to check that δ is the least improvement an exchange move can attain. Hence, the maximum number of moves needed to converge to a local optimum is

$$\frac{\gamma - \gamma_k^*}{\delta}, \tag{3}$$

where γ is bounded by the worst-case tour, that is, a tour traversing all the n vertices in the graph every time the robot moves from a pose to another. Being k the length of the tour, we have the following bound on the convergence complexity:

$$\frac{\gamma - \gamma_k^*}{\delta} \;\leq\; \frac{\Delta k n - \gamma_k^*}{\delta} \;\leq\; \frac{\Delta}{\delta} kn. \tag{4}$$

Consequently, assuming Δ and δ as constants, the complexity to converge for ExchangeProcedure(·) is $O(kn)$, which, along with the complexity of a single move, leads to an overall complexity of $O(k^3 n^2)$.

As a consequence, the complexity of our GRASP method is also $O(k^3 n^2)$, being ExchangeProcedure(·) the procedure with the highest complexity. Despite this worst-case complexity, in the next section we show that the GRASP method we propose is efficient in finding good quality solutions (that is, solutions whose cost is low) in randomly generated and in realistic grids.

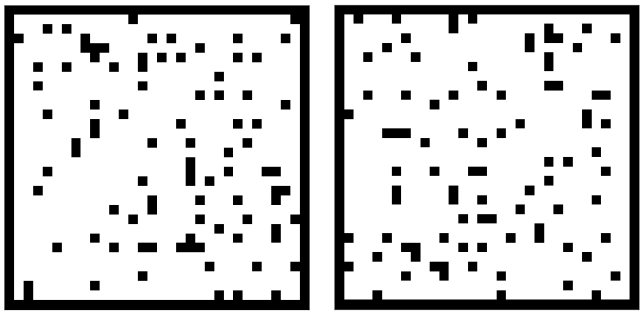

Figure 2: Two examples of randomly-generated grids. The density of free cells in the grids is about 90%.

Note that, as we discussed in the previous section, applying 2-OPT before ExchangeProcedure(·) (see Algorithm 3) has the effect to improve γ, and thus to reduce the gap $\gamma - \gamma_k^*$, mitigating the worst case complexity for the most critical phase of our GRASP method.

4.4 Global Optimum

One could wonder whether our GRASP method has a non-zero probability to reach a global optimum. It is easy to show that there always exists a value of the parameter L such that the GRASP we defined has a non-zero probability to find an optimal solution. This is equivalent to say that, for some value of L, GreedyRandomizedSolution(L) creates a solution that, passing through LocalSearch(·), becomes an optimal solution S^*. To show that this is always possible, let us proceed backward, starting from S^* and undoing the moves of LocalSearch(·): if the solution obtained S^*_{init} (the one given as input to the local search) can be generated with non-zero probability by GreedyRandomizedSolution(L), for some value of L, we proved the statement. This is always true, since, for $L = |V|$, the GreedyRandomizedSolution(L) can generate any possible (feasible) solution.

However, this result does not say anything about the *minimum* value of L such that a globally optimal solution can be found with non-zero probability. It turns out that such a value of L is strictly related to the specific problem instance. In Section 5 we discuss the tuning of L.

5. EXPERIMENTS

In this section we present the experimental results obtained on different environments. To validate our metaheuristic approach, we compare the costs of the solutions (time taken to complete the tour, composed of travelling and perception time) found by our GRASP method with those of the solutions obtained by the state-of-the-art algorithm presented in [3] and here called "cSPP+TSP" that, as discussed in Section 2, solves two independent subproblems. We implement our method in C++[1] (using uniform pseudo-random generation and samping) and we use the cSPP implementation in Matlab/Gurobi, as kindly provided by the authors of [3], along with a shelf TSP solver implementation. An unbiased comparison between computing times of our approach and those of cSPP+TSP is not straightforward, since, even leaving aside the Matlab wrapping, a significant part of the computational burden is delegated to commercial and fully-optimized solvers (i.e., Gurobi and Concorde). For this reason the comparison will mostly address the cost of the solutions found and only limitedly the effort spent to find them. Both approaches are compared on random grids of various sizes and on the grid discretization of a real indoor environment, taken from a public repository. In both cases, we consider different covering tool parameters (range r and angle θ). Moreover, we provide the computational times obtained by running our software on realistic-size instances. All the algorithms are run on a computer equipped with an Intel Core i7@2.70 GHz CPU and 16 GB RAM. In the experiments, we set the costs (time) for rotation movements (from $\alpha^{(i)}$ to $\alpha^{(i+1)}$ or vice versa), for straight movements (between adjacent cells), and for perception (function $T(\cdot)$) to 0.5, 1, and 1, respectively.

[1]Code is available upon request.

	$\theta = \pi$			
	$r = 10$		$r = 15$	
grid size	cSPP+TSP	GRASP	cSPP+TSP	GRASP
10×10	30.3 (3.9)	**20.4** (3.6)	35.7 (2.6)	**19.9** (1.7)
20×20	97.2 (4.6)	**75.4** (4.6)	91.9 (5.3)	**58.9** (3.6)
30×30	189.7 (8.2)	**160.3** (5.4)	179.4 (7.8)	**140.1** (4.0)
40×40	331.6 (7.7)	**289.2** (6.9)	299.6 (6.5)	**241.8** (8.9)

	$\theta = \frac{\pi}{2}$			
	$r = 10$		$r = 15$	
grid size	cSPP+TSP	GRASP	cSPP+TSP	GRASP
10×10	39.4 (3.2)	**24.9** (2.3)	42.5 (2.7)	**24.9** (2.1)
20×20	110.1 (4.6)	**96.0** (3.2)	104.7 (5.2)	**77.9** (3.2)
30×30	235.9 (7.8)	**219.4** (4.9)	214.7 (5.8)	**177.2** (5.2)
40×40	403.8 (10.3)	**378.3** (9.9)	346.8 (12.3)	**312.2** (7.3)

Table 1: Average costs (and 95%-confidence values) of the solutions found by our GRASP method and cSPP+TSP, for different grid sizes and values of r and θ. **Bold** indicates the best results.

5.1 Random Grids

Random grid environments are generated starting with a grid only composed of free cells and then randomly adding obstacle cells, guaranteeing the strong connection property (namely, each free cell must be reachable from all the other free cells considering 4-connectivity), until the percentage of free cells in the grid is about 90% of the total amount of cells. Figure 2 shows two examples of randomly generated grids. For each grid size, we create 10 different random grids and we compute the average cost of the solutions found. The range r of the sensor tool is set to 10 or 15 (cells) and the angle range θ can be $\frac{\pi}{2}$ or π.

After some preliminary experiments, we set the parameter $L = 16$, randomizing over the 16 locally-best vertices (see Section 4.1). An analysis of the impact of the parameter L is reported later. In our experiments, we stop the GRASP algorithm after 64 iterations (the stopping criterion of Algorithm 1).

Table 1 reports the results obtained with our GRASP method and with cSPP+TSP. The GRASP method always outperforms cSPP+TSP, especially when the number of poses needed to cover the environment tends to be small ($r = 15$

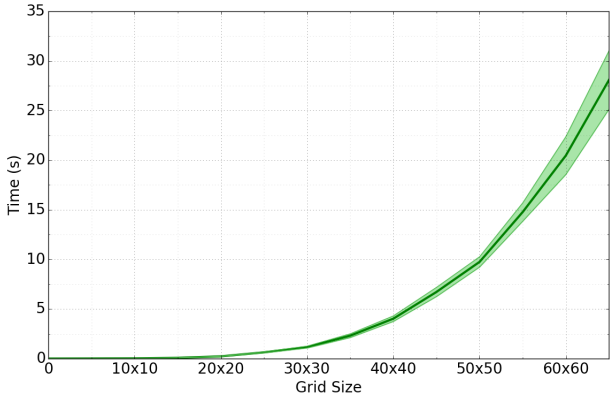

Figure 3: Average computing times (bold line) and 95%-confidence intervals (thin lines) of a single iteration of our GRASP method. The average computing times has been computed over 10 instances of each grid size in $\{5 \times 5, 10 \times 10, ..., 65 \times 65\}$.

and $\theta = \pi$). To explain this result, first note that, if we call A (one of) the minimum sets of perception poses needed to cover a given environment (that is, an art gallery solution found in step (a) of [3]), all the tours that visit less than $|A|$ poses violate the coverage constraint and thus cannot be solutions for the coverage problem. Increasing r and θ, the size of A decreases and the space of possible solution enlarges as all tours visiting more than $|A|$ poses are potential solutions (note also that there are no guarantees that a tour with few poses is shorter than a tour with more poses). This means that problems with small r and θ values tend to be more constrained and the TSP part of the cSPP+TSP tends to become predominant. Hence, from a coverage point of view, problem instances with large-range covering tools are intrinsically harder to solve with the two-step approach of [3].

The size, in terms of number k of poses, of the solution tours produced by our GRASP method is usually larger (about 50%) than that of the solution tours produced by cSPP+TSP. This means that, in general, our tours are overall faster but make more perceptions (consequently, greatly reducing the travel time). The reason is that, in solving the first subproblem of cSPP+TSP, the number of poses is minimized. An interesting observation is that this minimization is independent of the sensing time, namely the tours produced by cSPP+TSP have the same number of poses regardless of the sensing time. On the other hand, our method produces tours with a number of poses that depend on the time required to make a perception. If this time is small, as we have considered in our experiments, then the number of poses is relatively large. If the sensing time grows, then the number of poses returned by our method decreases, eventually approaching that returned by cSPP+TSP for very large sensing times.

Figure 3 shows the computing times for a single iteration of our GRASP method. Notice that the x-axis is quadratic w.r.t. the instance size n, so the curve looks much steeper than what actually is. The growth exhibits a cubic shape in the number of vertices of the movement graph and, although our not fully-optimized and research-oriented implementation, the single iteration times seem compatible with many practical applications (consider that paths are calculated offline). Also, the total time available to find a solution can be easily set by the user, who can tune the number of the GRASP iterations. One of the key aspects of GRASP is that, differently from cSPP+TSP, the amount of computational resources and the time available to solve the problem can be traded-off with the quality of the solution found. Indeed, one can increase the number of iterations and the L parameter obtaining better incumbent solutions, at the cost of an higher computing time. Moreover, GRASP can be easily speeded up by running iterations in parallel. For instance, using openMP in our C++ implementation, we computed 64 iterations for a 60×60, in less then 4 minutes, paralleliz-ing over 8 cores. The time our method takes to perform 64 iterations is comparable with that required by cSPP+TSP, as we found in our experiments and also reported in [3], even though this not constitute a rigorous comparison of the computational times, because of the different languages and environments, as already discussed above.

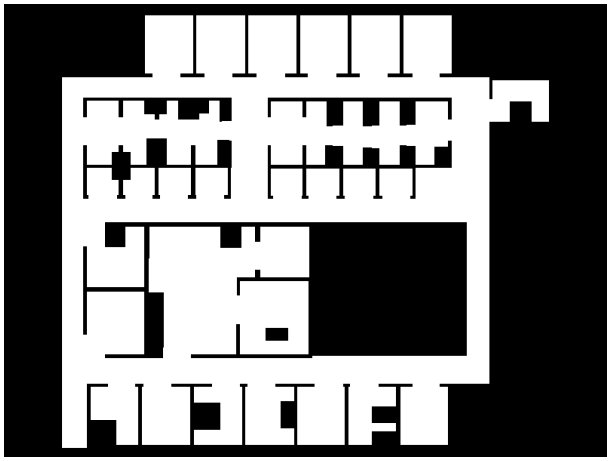

Figure 4: A real indoor environment map, from the Radish repository.

5.2 Real Environment

To further assess our metaheuristic approach we compare the costs of its solutions with those obtained by cSPP+TSP on a real indoor environment (see Figure 4), taken from the Radish repository[2]. A grid cell of our representation corresponds to 15×15 pixels of the original map image. A cell is free if all the corresponding pixels are white (clear), it is an obstacle otherwise. All the other settings are equal to those used for random grids.

The results shown in Figure 5 confirm those obtained on random grids, highlighting the improvement in the solution quality obtained by our approach, which becomes more evident as r and θ increase. Note that cSPP+TSP does not always return better solutions when r increases. This reinforces the idea that optimizing independently the two subproblems (a) and (b) (see Section 2) does not necessarily provide good solutions.

We use the real environment of Figure 4 as benchmark to evaluate how the costs of the solutions found by our approach change with the parameter L. For this purpose, we fix $r = 15$ and $\theta = \pi$ and we run 1000 iterations for

[2] `http://cres.usc.edu/radishrepository/view-one.php?name=sdr_site_b`

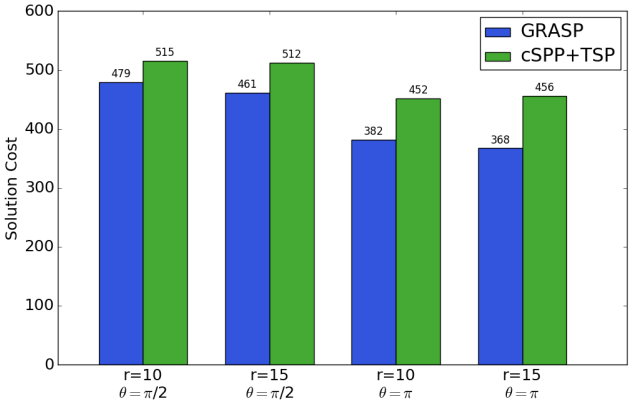

Figure 5: Costs of the solutions found by our GRASP method and cSPP+TSP, for different values of r and θ.

$L \in \{4, 8, 16, 32\}$. From the solutions found by all the iterations for a given L, we compute the empirical distribution $\hat{F}_{1000}(x)$ defined as the fraction of solutions found with cost less than or equal to x. The empirical distribution is shown in Figure 6 together with a zoomed version for small values of x (namely, for the high-quality solutions). An intuitive result is that, increasing the value of L, the average cost of the solutions increases due to randomization. Indeed, the mean costs we found for $L \in \{4, 8, 16, 32\}$ are 433.5, 433.6, 435.8, and 441.7, respectively. However, the probability of finding high-quality solutions can decrease with small and large values of L, as shown in Figure 6. Specifically, for $L = 4$, the GRASP method is too constrained by an almost-deterministic initial solution, which traps the algorithm in few local optima. On the contrary, for $L = 32$ the initial solution created by GreedyRandomizedSolution$(\cdot)$ is "too random" to quickly converge to good solutions. For this reason, $L = 8$ and $L = 16$ are the values that perform better, providing a good compromise between the time to converge to good solutions and the probability to find a high-quality solution. In fact, they have found the two solutions with the least cost in the environment of Figure 4. An open issue for further investigation is to find relationships between values of L and the problem setting (e.g., environment size and density of obstacles).

Figure 6 shows also the cost of the solution found by cSPP+TSP w.r.t. the cost distribution of the GRASP. It is worth to point out that, at the first iteration of the GRASP, the cost of the incumbent found is lower than that obtained by cSPP+TSP with a probability of about 75%, for $L = 8$. After only 5 iterations the same probability is $1 - 0.25^5 \sim 99.9\%$. This simple numerical example further highlights the reliability of our GRASP in terms of the cost of the solutions found.

6. CONCLUSIONS

In this paper, we presented a method based on the GRASP methodology for solving a version of the coverage problem in which an autonomous mobile robot has to sweep, with a tool of limited range and limited angular field of view, a grid environment. Our approach features two phases, which are iteratively performed: one in which an initial feasible covering tour is randomly created and another one in which this tour is improved by local modifications. Experiments

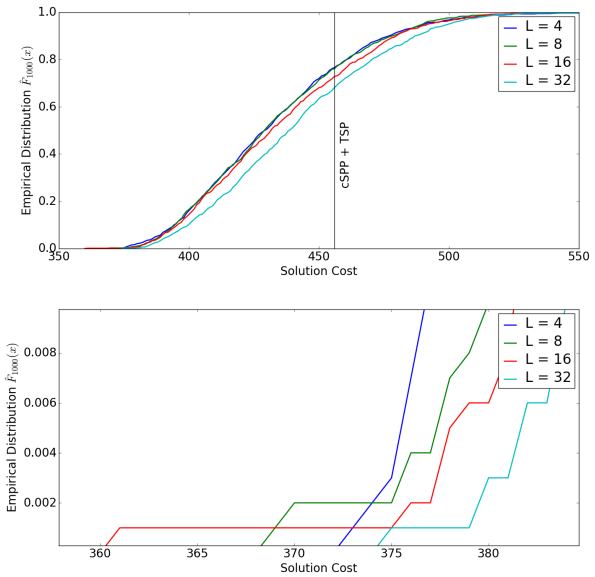

Figure 6: Empirical distribution $\hat{F}_{1000}(x)$ computed over 1000 samples (iterations) for $L \in \{4, 8, 16, 32\}$ (top) and a zoomed portion of the distribution in the region of high-quality solutions (bottom).

in randomly generated grids and in a grid representing a real environment show that the proposed method consistently outperforms a state-of-the-art method for covering grids with limited-footprint tools.

In the future, it would be interesting to extend our method to environments that are discretized with non-regular patterns, for example according to the footprint of the covering tool and to the presence of obstacles. Moreover, a more detailed analysis of the possible bounds on the difference between the quality of the covering tour returned by our method and that of the optimal covering tour (which is in general unknown) could be carried out. Finally, incorporating uncertainty (e.g., of localization and locomotion) in the method and extending it to multirobot settings represent promising research directions.

Acknowledgements

The authors gladly thank Asif Arain for providing the code of the cSPP+TSP method used for experimental comparison.

REFERENCES

[1] N. Agmon, N. Hazon, and G. Kaminka. Constructing spanning trees for efficient multi-robot coverage. In *Proc. ICRA*, pages 1698–1703, 2006.

[2] F. Amigoni. Experimental evaluation of some exploration strategies for mobile robots. In *Proc. ICRA*, pages 2818–2823, 2008.

[3] A. Arain, M. Trincavelli, M. Cirillo, E. Schaffernicht, and A. Lilienthal. Global coverage measurement planning strategies for mobile robots equipped with a remote gas sensor. *Sensors*, 15(3):6845–6871, 2015.

[4] N. Basilico and F. Amigoni. Exploration strategies based on multi-criteria decision making for searching

environments in rescue operations. *Autonomous Robots*, 31(4):401–417, 2011.

[5] M. Bellmore and G. Nemhauser. The traveling salesman problem: a survey. *Operations Research*, 16(3):538–558, 1968.

[6] Y. Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *Proc. IAT*, pages 302–308, 2004.

[7] W.-P. Chin and S. Ntafos. Optimum watchman routes. In *Proc. SOCG*, pages 24–33, 1986.

[8] H. Choset. Coverage for robotics–a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1-4):113–126, 2001.

[9] T. Danner and L. Kavraki. Randomized planning for short inspection paths. In *Proc. ICRA*, pages 971–976, 2000.

[10] A. Dumitrescu and C. Tóth. Watchman tours for polygons with holes. *Computational Geometry*, 45(7):326–333, 2012.

[11] J. Faigl, M. Kulich, and L. Přeučil. A sensor placement algorithm for a mobile robot inspection planning. *Journal of Intelligent & Robotic Systems*, 62(3-4):329–353, 2011.

[12] P. Fazli, A. Davoodi, P. Pasquier, and A. Mackworth. Complete and robust cooperative robot area coverage with limited range. In *Proc. IROS*, pages 5577–5582, 2010.

[13] T. Feo and M. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[14] Y. Gabriely and E. Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Computational Geometry*, 24(3):197–224, 2003.

[15] E. Galceran and M. Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276, 2013.

[16] A. Janani, L. Alboul, and J. Penders. Multi-agent cooperative area coverage: Case study ploughing (extended abstract). In *Proc. AAMAS*, pages 1397–1398, 2016.

[17] A. Kroll, W. Baetz, and D. Peretzki. On autonomous detection of pressured air and gas leaks using passive ir-thermography for mobile robot application. In *Proc. ICRA*, pages 921–926, 2009.

[18] J. Mitchell. Approximating watchman routes. In *Proc. SODA*, pages 844–855, 2013.

[19] M. Resende. Greedy randomized adaptive search procedures greedy randomized adaptive search procedures. *Encyclopedia of Optimization*, pages 1460–1469, 2009.

[20] Y. Tomioka and H. Kitazawa. Collaborative patrol planning of mobile surveillance cameras for perfect observation of moving objects. In *Proc. ICME*, pages 1–6, 2013.

[21] Y. Tomioka, A. Takara, and H. Kitazawa. Generation of an optimum patrol course for mobile surveillance camera. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(2):216–224, 2012.

[22] R. Yehoshua, N. Agmon, and G. Kaminka. Robotic adversarial coverage of known environments. *The International Journal of Robotics Research*, 35(12):1419–1444, 2016.

[23] A. Zelinsky, R. Jarvis, J. Byrne, and S. Yuta. Planning paths of complete coverage of an unstructured environment by a mobile robot. In *Proc. ICAR*, pages 533–538, 1993.