Intl. Trans. in Op. Res. xx (2020) 1–22

Multi-robot search for a stationary object placed in a known environment

Miroslav Kulich*, Libor Přeučil

Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, Jugoslávských partyzánů 1580/3, 160 00 Prague 6, Czech Republic
E-mail: kulich@cvut.cz,preucil@cvut.cz

Abstract

The paper addresses the problem of multi-robot search for a stationary object in a priori known environment. Two variants of the problem are studied given the working environment represented by a graph. The first one is an extension of the Traveling Deliveryman Problem for multiple vehicles, while the second one is a generalization of the Graph Search Problem. A novel algorithm is presented to solve both these problems, which is based on a combination of Greedy Randomized Adaptive Search Procedure with Variable Neighborhood Descent. A set of experimental evaluations was conducted over the benchmark instances derived from the TSPLIB library. The results obtained show that the proposed approach outperforms state of the art approaches in quality of results for both problems. Moreover, computational times for problems with the size of few hundreds of vertices allow using the approach for on-line decision making in search and rescue scenarios.

Keywords: Combinatorial optimization; Metaheuristics; Graph Search Problem; Traveling Deliveryman Problem

1. Introduction

Assume a mobile robot autonomously operating in a priori known environment, in which a stationary object of interest is randomly placed. The objective is to find the object, whose position is not known in advance, in minimal time.

This problem is formulated as the Traveling Deliveryman Problem (TDP) provided that the environment is represented by a graph and probability of appearance of the object is the same for all vertices in the graph. TDP is well-known problem in the operational research community, and it has been studied from various perspectives during the last few years. Although the problem is NP-hard Afrati et al. (1986), several authors introduced exact algorithms (Fischetti et al., 1993; Gouveia and Voß, 1995; Méndez-Díaz et al., 2008). Besides this, Integer Linear Programming with Branch-and-Cut and Branch-Cut-and-Price approaches were proposed for the Time-Dependent Traveling Salesman Problem which

* Corresponding Author: kulich@cvut.cz

generalizes TDP (Abeledo et al., 2012; Miranda-Bront et al., 2010, 2013; Godinho et al., 2014). The best exact algorithm Abeledo et al. (2012), nevertheless, can solve instances with up to 107 vertices to optimality in several hours.

More useful are heuristics and meta-heuristics which provide good quality solutions with much lower computational effort. These rely particularly on Greedy Randomized Adaptive Search Procedure (GRASP), introduced originally in Feo and Resende (1995), and Variable Neighborhood Search (VNS), proposed in Hansen and Mladenović (1997). Salehipour et al. (2011) employ a GRASP for TDP and compare the impact of VNS procedure as a local search phase with Variable Neighborhood Descent. Mladenović et al. (2012) propose a General VNS (GVNS), which improves Salehipour's results. Further improvements were achieved by Silva et al. (2012) who propose a simple multi-start heuristics combined with an Iterated Local Search procedure. To the best of our knowledge, the approach by Silva is thus the one producing the best results. Nevertheless, a time needed to compute problems containing 100 vertices is more than ten seconds and instances with 200 vertices are computed in approx. one minute.

Approaches used by the robotic community are simpler. Sarmiento et al. (2004) propose a modification of breadth-first algorithm which iteratively constructs all possible routes of the defined length, fixes the most promising one and starts the next search from this route as a prefix. Finally, a modified depth-first search algorithm with pruning and limited branching was introduced in our previous work (Kulich et al., 2014).

The Graph Search Problem (GSP), introduced in Koutsoupias et al. (1996), is formulated under the same settings as TDP, with the only difference that each vertex has assigned a probability of finding the object when visiting the vertex and probabilities of the vertices differ in general. Besides some theoretical results regarding approximation schemes presented in Ausiello et al. (2000), no further developments are present in the related literature. The only exception is our recent work (Kulich et al., 2016) in which a tailored GRASP meta-heuristics for the GSP is introduced which finds near-optimal solutions for TDP and GSP problems up to 107 vertices in about one second of computing time.

Other variants of the TDP received some attention in the last few years from the operational research community also. Regarding the single-vehicle case, Heilporn et al. (2010) address the TDP with time windows (TDPTW), where each vertex must be visited within a particular interval of time. They propose different Integer Linear Programming (ILP) formulations and develop exact Branch-and-Cut algorithms, in addition to some heuristic approaches. A different variant is considered in Dewilde et al. (2013), named the TDP with profits, where a time-dependent (decreasing) profit is collected when visiting a particular vertex. The whole set of vertices is not required to be visited, and the objective is to maximize the overall profit of the tour. The authors consider a Tabu Search metaheuristic approach and report good quality results, requiring on average more than 1000 seconds for instances having 200 or more vertices. Finally, we point out the work by Rivera et al. (2016) where the vehicle is assumed to have a limited capacity and multi trips are allowed. They propose two different ILP formulations and a graph transformation procedure which allows tackling the problem by solving a Shortest-Path problem with resource constraints. Computational results are reported on classical Capacitated Vehicle Routing Problem (CVRP) instances.

Within the multi-vehicle setting, which this paper primarily focuses, several exact and heuristic algorithms have been proposed recently. Lysgaard and Wøhlk (2014) propose a Branch-and-Cut-and-Price (BPC) for the Cumulative Capacitated Vehicle Routing Problem (CCVRP), which is the natural extension of the single-vehicle case where the vehicles have finite capacity. Similarly, Luo et al. (2014)

study the problem where the capacity of the vehicles is not binding, but the routes must not exceed a predefined maximum distance. Computational results are reported for instances with up to 50 vertices, showing that the problem is indeed difficult to be approached by exact algorithms. The CCVRP has also been addressed by heuristic approaches. Ngueveu et al. (2010) propose two Memetic Algorithms (MA) and evaluate the approaches on CVRP instances ranging from 50 to 420 vertices. The results obtained are later improved in Mattos Ribeiro and Laporte (2012), which propose an Adaptative Large Neighborhood Search (ALNS). In a follow-up paper, Ke and Feng (2013) tackle the CCVRP by a two-phase metaheuristics which can improve the best-known solution in 9 of the instances considered in the former papers. Finally, we mention the application considered in Ribeiro et al. (2012), where a problem related to the operations within oil fields is solved. The structure of the objective function is similar to the one in the GDP, but they consider a multi-vehicle setting and time-windows at the vertices, generalizing, therefore, the TDPTW. They compare three different metaheuristics on instances 50, 100 and 500 vertices and allowing the methods to execute 30, 180, and 7200 seconds, respectively.

Some inspiration can be also found in approaches to the Multiple Traveling Salesman Problem (mTSP) or other routing problems. Bektas (2006) provides a review of ILP formulations as well as regarding heuristics approaches. Besides genetic algorithms, neural networks, or ant colony optimization, the cluster-first route-second scheme plays an important role. The fundamental idea of this approach is to split all vertices into M clusters based on their location in space (M is the number of salesmen) and solve the traditional Traveling Salesman Problem for each cluster separately. For example, Sathyan et al. (2015) use k-means clustering for the first phase followed by application of a genetic algorithm or 2-opt heuristics. Boone et al. (2015) employ an initial k-means clustering and modify it by taking points from the cluster with the largest tour distance and adding them to one of the smaller clusters. Convex hulls, fuzzy logic, and the TSP solution are used to determine which points to switch. Geetha et al. (2009) improved the k-means algorithm for the Capacitated Clustering Problem by incorporating a priority measure to the criterion on which are vertices assigned to clusters.

In our recent paper (Kulich et al., 2017) we presented a cluster-first route-second approach for the Multiple Traveling Deliveryman Problem (mTDP) and the multi-vehicle case of GSP (mGSP). The approach extended our GRASP-based meta-heuristic for the single-vehicle problems Kulich et al. (2016) by incorporating the clustering phase respecting special aspects of mTDP and mGSP.

In this paper, we build on experience from our recent work, especially Kulich et al. (2016, 2017) and introduce a novel approach based on GRASP and VND for mTDP and mGSP. The proposed approach uses a tailored clustering from Kulich et al. (2017) in GRASP as a constructive heuristics for generation of initial solutions together with a set of designed operators used in VND to improve these solutions. Furthermore, we derive equations for fast evaluation of changes of an objective function caused by application of particular operators and show that neighborhoods defined by these operators can be explored in time $O(n^2)$, where n is a number of vertices, which is the same as the complexity of operators for TSP.

The rest of the paper is organized as follows. The problem is defined in Section 2, while the proposed approach for mTDP and mGSP is introduced in Section 3. Computational results including instances of both mTDP and mGSP as well as experiments with real robots are presented and discussed in 4. Finally, Section 5 is dedicated to concluding remarks and future directions.

2. Problem Formulation

The formulation of the Graph Search Problem for multiple vehicles is a direct extension of the single-vehicle case as presented e.g. in Kulich et al. (2014, 2016). That is, formally, given

- a complete undirected graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_N\}$ stands for a finite set of N vertices and E is a set of edges between these vertices: $E = \{e_1, e_2, \ldots e_{N^2}\}$, $e_{ij} = (v_i, v_j)$, $v_i, v_j \in V$, $i \neq j$,
- $d: E \to \mathbb{R}$: a cost d_{ij} associated with each edge e_{ij} representing a length of the shortest path from i to j,
- $w: V \to \langle 0, 1\rangle$: a weight for each vertex representing a probability of a presence of the searched object at the vertex,
- the number of the vehicles M, and
- $s_i \in V, i \in \langle 1, M\rangle$: starting vertices of the vehicles (note that several vehicles can start from the same vertex in general),

define a *route* $\boldsymbol{x} = \langle x_1, x_2, \ldots x_k\rangle$ as a sequence of vertices of G, i.e. $x_i \in V$ for $i \in \langle 1, k\rangle$. The overall objective is then to find a tuple of M routes $\boldsymbol{X} = \langle \boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^M\rangle$ that

1. visits all vertices of V at least once:

$$\forall v \in V \;\; \exists i, j : x_j^i \in \boldsymbol{x}^i \text{ and } x_j^i = v,$$

2. all the routes start at the given starting vertices:

$$\forall i \in \langle 1, M\rangle : x_1^i = s_i \text{ and}$$

.

3. minimizes the expected time to find the object[1]:

$$\boldsymbol{X} = \underset{\boldsymbol{\chi}\in\boldsymbol{\mathcal{X}}}{\operatorname{argmin}}\, \mathbb{E}(T|\boldsymbol{\chi}) = \sum_{i=1}^{M}\sum_{j=1}^{|\boldsymbol{\chi}^i|}\delta^{\boldsymbol{\chi}^i}(j) w^{\boldsymbol{\chi}^i}(j), \tag{1}$$

where

$$\delta^{\boldsymbol{\chi}^i}(j) = \sum_{k=1}^{j-1} d(x_k^i x_{k+1}^i) \tag{2}$$

is the time when the vertex x_j^i, the j-th vertex of the i-th route $\boldsymbol{\chi}^i \in \chi$ is visited, $w^{\boldsymbol{\chi}^i}(j)$ is the weight of x_j^i, and $\boldsymbol{\mathcal{X}}$ is a set of all possible sets of routes in G. The minimal expected time is then

$$T_{exp} = \mathbb{E}(T|\boldsymbol{X}) = \sum_{i=1}^{M}\sum_{j=1}^{|\boldsymbol{x}^i|}\delta^{\boldsymbol{x}^i}(j) w^{\boldsymbol{x}^i}(j). \tag{3}$$

[1] We assume that vehicles move with a constant velocity and thus time is proportional to distance.

Shortly, the aim is to minimize the average time the vertices are visited weighted by probabilities assigned to the vertices.

The multi-vehicle Traveling Deliveryman Problem is a particular variant of mGSP with the only difference that the probability of finding the object is the same for all the vertices in the graph. These probabilities can be omitted from the equations for this case.

3. The approach

The proposed approach to mGSP is based on the combination of the Greedy Randomized Adaptive Search Procedure (GRASP) (Feo and Resende, 1995) and Variable Neighborhood Descent (VND) (Ribeiro and Vianna, 2005), see the general schema in Algorithm 1. The meta-heuristics consecutively constructs initial solutions using various heuristics from a given set of heuristics H (line 4). Each of the obtained solutions is improved by a sequence of local search steps (line 5) and its cost is updated accordingly (line 6). If the improved solution is better than the current best solution, which is initially set to a high number in the algorithm (line 1), the best solution and its cost are updated (lines 7–9). After all initial solutions are processed, the best solution found is reported (line 10). N_{it} at line 3 is a predefined number of initial solutions generated by a single heuristics. More detailed description of the particular steps of the GRASP follows in the next paragraphs.

Algorithm 1: GRASP scheme.

```
1  z_best ← ∞.
2  foreach h ∈ H do
3      for k = 1, ..., N_it do
4          Obtain a feasible solution X using h.
5          Improve X by applying a local search step
6          Update cost z of X.
7          if z < z_best then
8              X_best ← X
9              z_best ← z
10 return X_best
```

3.1. Initial solution generation

Generation of initial solutions follows ideas of the clustering approach introduced in Kulich et al. (2017). Because the algorithm in Kulich et al. (2017) is deterministic and a set of different initial solutions has to be generated for GRASP, it was randomized as shown in Algorithm 2.

The approach, which has a greedy nature, initializes particular routes first so that each route contains just the start vertex of the corresponding vehicle. Moreover, the time needed to traverse each route is set to zero (lines 1–3). The algorithm maintains the set of not yet assigned vertices, R, which is initialized

at line 4. Vertices from R are examined and consecutively attached to most appropriate routes in the loop (5–12). This is done by processing each pair consisting of a vertex $v \in R$ and a route $\chi \in X$: time needed to visit the v if it is attached to the end of χ is determined (line 8) and used to compute a cost of this attachment (line 9). We employ two different cost functions defining two heuristics for GRASP. The first one is simply the distance as computed at line 8, while the second one is defined as

$$c_{\chi,v} = \frac{d}{1 + w_v},$$

which prefers vertices with high weights.

Contrary to the deterministic approach, where the pair (χ, v) with the lowest cost is selected, a pair is chosen randomly according to the uniform distribution from a pool of κ pairs with the lowest costs (line 10). Finally, v is attached to χ (line 11), removed from R (line 12), and time to traverse χ is updated (line 13). The algorithm finishes when R is empty and $\boldsymbol{\mathcal{X}}$ is reported.

Algorithm 2: Generation of an initial solution.

Input: M – the number of vehicles
$G = (V, E)$ – a graph
d_{ij} – costs of edges
w_i – probabilities associated with vertices
$s_i \in V, i \in \langle 1, M \rangle$ – starting vertices of vehicles
Output: $\boldsymbol{\mathcal{X}} = \langle \chi^1, \chi^2, \ldots \chi^M \rangle$ – a tuple of sequences representing particular routes

```
1  for i ← 1 to M do
2      χ^i ← ⟨s_i⟩
3      δ^{χ^i} ← 0
4  R ← V \ {s_1, s_2, ..., s_M}
5  while R ≠ ∅ do
6      foreach χ ∈ 𝒳 do
7          foreach v ∈ R do
8              d = δ^χ + d(v, χ_|χ|), where
                   χ_|χ| is the last vertex of the route χ
9              c_{χ,v} = c(d, w_v)
10     select (χ, v) according to c_{χ,v}
11     χ ← χ + v
12     R ← R \ {v}
13     δ^χ ← δ^χ + d(v, χ_|χ|)
14 return 𝒳 = ⟨χ^1, χ^2, ... χ^M⟩
```

3.2. Route improvement

Each route generated in Algorithm 2 is improved using Variable Neighborhood Descent as depicted at line 5 of Algorithm 1. VND is a local search procedure which iteratively and systematically explores several neighborhoods of a given solution and updates the solution to the best neighbor found. Assume for example 2-opt operator which takes two non-adjacent edges and replaces them by two new edges as shown in Fig. 1. A neighborhood of a route χ defined by this operator is a set of all routes originating from χ by application of some 2-opt operation.

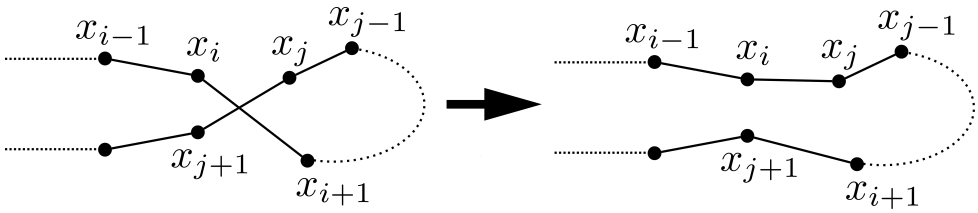


Fig. 1: *2-opt* operation: edges (x_i, x_{i+1}) and (x_j, x_{j+1}) are removed and replaced by edges (x_i, x_j) and (x_{i+1}, x_{j+1}).

A general structure of VND is depicted in Algorithm 3. The algorithm consecutively takes operators from a given set of local search operators O (lines 2–8), where $\mathcal{N}_{O_i}(\boldsymbol{x})$ stands for a neighborhood of $\boldsymbol{x}$ and $f(\boldsymbol{\chi})$ is a cost of a solution $\boldsymbol{\chi}$. If a better solution than the current one is found in a neighborhood defined by the current operator, it becomes a new solution (line 5) and the process is restarted from the first operator in the set (line 6). Otherwise, the algorithm continues with a next operator (line 8). The algorithm finishes when all operators are processed without improvement. The found solution is reported in that case (line 9).

Algorithm 3: General structure of Variable Neighborhood Descent

Input: $\boldsymbol{x}$ – an initial solution
$O = \langle o_1, o_2, \ldots, o_n \rangle$ – a sequence of local search operators
Output: $\boldsymbol{x}$ – an improved solution

```
1 i ← 1
2 while i ≤ n do
3     x' ← argmin_{χ∈N_{O_i}(x)} f(χ)
4     if f(x') < f(x) then
5         x ← x'
6         i ← 1
7     else
8         i ← i + 1
9 return x
```

In the proposed approach, we consider two local search operators working with a single route:

- ***2-opt:*** as described above and
- ***Swap:*** select two vertices in the tour and interchange them

together with three operators involving two routes:

- ***Move:*** remove a vertex from one route and add it to the second route,
- ***Exchange:*** select two vertices from different routes and exchange them, and
- ***Swap tails:*** select tails of two different routes and exchange them.

These operators, which are detailed in Section 3.2.1, define the improvement process as depicted in Algorithm 4. Each route of an initial solution is improved by VND which employs *2-opt* and *Swap* (lines 1-2) at first. The improved solution is then refined by another VND with *Move*, *Exchange*, and *Swap tails* operators (lines 3). Whenever one of these operators is successful, VND with *2-opt* and *Swap* is called again for both involved routes. Each route of the solution is finally refined by so-called intensification and the result is reported.

Algorithm 4: Solution improvement.

Input: $\boldsymbol{X}$ – an initial solution
Output: $\boldsymbol{X}$ – an improved solution

1 **foreach** $\boldsymbol{x} \in \boldsymbol{X}$ **do**
2 $\quad \boldsymbol{x} \leftarrow VND(\boldsymbol{x}, \langle$*2-opt, Swap*$\rangle)$
3 $\boldsymbol{X} \leftarrow VND(\boldsymbol{X}, \langle$*Move, Exchange,Swap tails*$\rangle)$
4 **foreach** $\boldsymbol{x} \in \boldsymbol{X}$ **do**
5 $\quad \boldsymbol{x} \leftarrow \text{intens}(\boldsymbol{x}, 0, \emptyset, x_1)$
6 **return** $\boldsymbol{X}$

Intensification is inspired by an adaptation procedure used in the Lin-Kernighan heuristics to solve the Traveling Salesman Problem Lin and Kernighan (1973). The procedure is recursive with limited backtracking. The idea is to consider each edge on the route sequentially as a seed for an improvement procedure which attempts to find an improved route by application of a sequence of 2-opt moves. Particular *2-opt* moves, in contrast to VND, do not necessarily improve the solution. Instead, a gain of the whole sequence is tracked: whenever a better solution is found, it is accepted as the new initial solution and the whole procedure is restarted from the first edge.

The procedure is shown in Algorithm 5. If a recursion depth is not reached (line 1), all vertices in the neighborhood $\mathcal{N}_{x_i}^{depth}$ of the current vertex x_i are sequentially processed. $\mathcal{N}_{x_i}^{depth}$ contains first n_{depth} vertices nearest to x_i according to distances d as defined earlier, where n_{depth} is a given size of the neighborhood dependent on current depth. If the neighbor has not been considered in the current iteration and the edge (x_k, x_{k+1}) is not next to (x_i, x_{i+1}), *2-opt* on these two edges is performed (line 4). If this leads to improvement of the current route, the process is restarted with the improved route as an initial solution (line 6). Otherwise, a recursive step is called with the next vertex (line 6). This process is repeated until all nodes have been considered as a seed.

Intensification is computationally demanding and it is therefore performed only for most promising solutions, i.e. it is applied only if the cost of the current solution is less than 110% of the current best solution.

Algorithm 5: Intensification procedure: $\text{intens}(\boldsymbol{x}, depth, R, x_i)$.

```
if depth < α then
    foreach x_k ∈ N_{x_i}^{depth} do
        if x_k ∉ R ∧ x_{k+1} ≠ x_i then
            x' ← 2-opt({x_i, x_{i+1}}, {x_k, x_{k+1}})
            if f(x') < f(x) then
                return intens(x', 0, ∅, x_1)
            else
                return intens(x', depth + 1, R ∪ {x_k}, x_k)
return x
```

3.2.1. Local search operators

Contrary to the Traveling Salesman Problem, where a change of solution cost caused by application of a local search operator (e.g. *2-opt*) can be straightforwardly determined in constant time, some preprocessing and additional data structures are needed for mGSP. Derivation of cost change for operators used in the improvement phase of GRASP follows. Note a property directly resulting from Eq. 1 – the expected time for all vehicles is a sum of the expected times for particular routes. This means that improving a route of one vehicle leads to the same improvement of the whole solution.

2-opt

Given a route $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_n \rangle$, its cost, which corresponds to the expected time of finding an object, is computed as:

$$\begin{aligned} c(\boldsymbol{x}) = \mathbb{E}(T|\boldsymbol{x}) = \delta_1 w_1 + \delta_2 w_2 + \cdots + \delta_i w_i \\ + \cdots + \delta_j w_j + \cdots + \delta_n w_n, \end{aligned} \tag{4}$$

where δ_i is time needed to reach the i-th vertex on the route $\boldsymbol{x}$ as defined in Eq. 2. Application of the *2-opt* operator on $\boldsymbol{x}$ and edges (x_i, x_{i+1}) and (x_j, x_{j+1}) as depicted in Fig. 1 results in the route $\boldsymbol{x}' = \langle x_1, x_2, \ldots, \ x_i, x_j, x_{j-1}, \ldots, x_{i+2}, x_{i+1}, x_{j+1}, \ldots, x_n \rangle$. Summing contribution of particular nodes in

the order they appear in $\boldsymbol{x}'$, its cost can be expressed as:

$$
\begin{aligned}
c(\boldsymbol{x}') = {} & \delta_1 w_1 + \delta_2 w_2 + \cdots + \delta_{i-1} w_{i-1} + \delta_i w_i \\
& + (\delta_i + d_{i,j})\, w_j \\
& + (\delta_i + d_{i,j} + d_{j,j-1})\, w_{j-1} + \ldots \\
& + (\delta_i + d_{i,j} + d_{j,j-1} \ldots d_{j-k+1,j-k})\, w_{j-k} + \ldots \\
& + (\delta_i + d_{i,j} + d_{j,j-1} \cdots + d_{i+2,i+1})\, w_{i+1} \\
& + (\delta_{j+1} - d_{i,i+1} - d_{j,j+1} + d_{i,j} + d_{i+1,j+1})\, w_{j+1} \\
& + \cdots + (\delta_n - d_{i,i+1} - d_{j,j+1} + d_{i,j} + d_{i+1,j+1})\, w_n
\end{aligned}
\tag{5}
$$

To compute improvement obtained by application of the operation (i.e. a difference of $c(\boldsymbol{x}') - c(\boldsymbol{x})$), the similar trick as in Mladenović et al. (2012) is used. The cost functions are summed first followed by subtraction of a double of $c(\boldsymbol{x})$:

$$
\begin{aligned}
& \Delta_{2-opt}(x, i, j) = \\
& \quad c(\boldsymbol{x}') - c(\boldsymbol{x}) = c(\boldsymbol{x}') + c(\boldsymbol{x}) - 2S_b(n) = \\
& \quad 2S_b(i) + (\delta_i + \delta_j + d_{i,j})(\gamma_j - \gamma_i) \\
& \quad + 2S_e(j+1) + (d_{i,j} + d_{i+1,j+1} - d_{i,i+1} - d_{j,j+1})(\gamma_n - \gamma_j) \\
& \quad - 2S_b(n),
\end{aligned}
\tag{6}
$$

where

$$
S_b(i) = \sum_{k=1}^{i} \delta_k w_k, \quad S_e(i) = \sum_{k=i}^{n} \delta_k w_k \tag{7}
$$

are contributions to the cost of first and last i vertices respectively. Note that $c(\boldsymbol{x}) = S_b(n) = S_e(1)$. γ_i is then defined as

$$
\gamma_i = \sum_{k=1}^{i} w_k. \tag{8}
$$

If S_b, S_e, and γ of all vertices are precomputed for a given $\boldsymbol{x}$, which can be done iteratively in $O(n)$, the complete 2-opt neighborhood is explored in $O(n^2)$. This is possible because a size of the neighborhood is n^2 and evaluation of a single 2-opt operation takes constant time thanks to the knowledge of all terms in Eq. 6. Algorithm 3 thus assumes that S_b's, S_e's, and γ's are precomputed and when a better solution is found, these values are updated at line 5.

Swap

Given a route $\boldsymbol{x}$ with a cost defined in Eq. 4, *Swap* interchanges vertices at positions i and j. The

resulting route is thus $\boldsymbol{x}' = \langle x_1, x_2, \ldots, x_{i-1}, x_j, x_{i+1}, \ldots, x_{j-1}, \;\; x_i, x_{j+1} \ldots, x_n \rangle$ and its cost

$$\begin{aligned} c(\boldsymbol{x}') &= S_b(i-1) + w_j(\delta_i + \Lambda_1) \\ &+ w_{i+1}(\delta_{i+1} + \Lambda_2) \cdots + w_{j-1}(\delta_{j-1} + \Lambda_2) \\ &+ w_i(\delta_j + \Lambda_3) \\ &+ w_{j+1}(\delta_{j+1} + \Lambda_4) + \cdots + w_n(\delta_n + \Lambda_4), \text{ where} \end{aligned} \tag{9}$$

$$\begin{aligned} \Lambda_1 &= d_{i-1,j} - d_{i-1,i} \\ \Lambda_2 &= \Lambda_1 + d_{j,i+1} - d_{j,i+1} \\ \Lambda_3 &= \Lambda_2 + d_{j-1,i} - d_{j-1,j} \\ \Lambda_4 &= \Lambda_3 + d_{i,j+1} - d_{i,j+1} \end{aligned} \tag{10}$$

Improvement after application of the operator is computed directly as a difference of the costs

$$\begin{aligned} \Delta_{swap}(x, i, j) &= c(\boldsymbol{x}') - c(\boldsymbol{x}) = w_j\Lambda_1 + (\gamma_{j-1} - \gamma_i)\Lambda_2 \\ &+ w_i\Lambda_3 + (\gamma_{n-1} - \gamma_j)\Lambda_4 + (\delta_j - \delta_i)(w_i - w_j). \end{aligned} \tag{11}$$

Again, all the terms in Eqs. 9 and 10 are known and the whole neighborhood $\mathcal{N}_{Swap}(\boldsymbol{x})$ can be thus explored in $O(n^2)$.

Move

The *Move* operation involves two routes. It removes the i-th vertex from a route $\boldsymbol{x}$ and adds it before j-th vertex of a route $\boldsymbol{y} = \langle y_1, y_2, \ldots, y_n \rangle$, so that resulting routes will be $\boldsymbol{x}' = \langle x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n, \rangle$ and $\boldsymbol{y}' = \langle y_1, y_2, \ldots, y_{j-1}, x_i, y_j, \ldots, y_n \rangle$. Influence of vertex removal is expressed directly as

$$\begin{aligned} &\Delta_{remove}(\boldsymbol{x}, i) = \\ &- w_i\delta_i + (\gamma_n - \gamma_i)(d_{i-1,i+1} - d_{i-1,i} - d_{i,i+1}) \end{aligned} \tag{12}$$

while cost change caused by the addition of a vertex is

$$\begin{aligned} \Delta_{add}(\boldsymbol{y}, j, p) &= w_p(\delta_{j-1} + d_{j-1,p}) \\ &+ (\gamma_n - \gamma_{j-1})(d_{j-1,p} + d_{p,j} - d_{j-1,i}), \end{aligned} \tag{13}$$

where p is the index of the vertex added. Improvement after application of the operation is finally

$$\Delta_{move}(x, i, y, j) = \Delta_{remove}(\boldsymbol{x}, i) + \Delta_{add}(\boldsymbol{y}, i, idx(x_i))$$

which can be determined in constant time assuming that γ's are precomputed. $idx(x_i)$ is the index of x_i. Note that all possible pairs of routes and all possible positions on them are evaluated resulting in a total complexity of neighborhood exploration $O(n^2)$.

Exchange

Exchange swaps i-th vertex of $\boldsymbol{x}$ with j-th vertex of $\boldsymbol{y}$. The resulting routes after application of Exchange will be $\boldsymbol{x}' = \langle x_1, x_2, \ldots, x_{i-1}, y_j, x_{i+1}, \ldots, x_n, \rangle$ and $\boldsymbol{y}' = \langle y_1, y_2, \ldots, y_{j-1}, x_i, y_j, \ldots, y_n \rangle$. Improvement of this operation can be computed as a sum of improvements of two Move operations, which can be further simplified. For the route $\boldsymbol{x}$ we get

$$\begin{aligned} \Delta_{ex}(\boldsymbol{x}, i, ,j) = &-w_i\delta_i + w_p(\delta_{i-1} + d_{i-1,p}) \\ &+ (\gamma_n - \gamma_i)(d_{i-1,p} + d_{p,i+1} - d_{i-1,i} - d_{i,i+1}) \end{aligned} \tag{14}$$

The same applies for $\boldsymbol{y}$ with reordered parameters. Improvement of the whole operation is then evaluable in constant time:

$$\Delta_{exchng}(\boldsymbol{x}, i, \boldsymbol{y}, j) = \Delta_{ex}(\boldsymbol{x}, i, \boldsymbol{y}, j) + \Delta_{ex}(\boldsymbol{y}, j, \boldsymbol{x}, i)$$

Swap tails

Finally, *Swap tails* takes the last part of $\boldsymbol{x}$ starting from the vertex i and swaps it with the tail of $\boldsymbol{y}$ starting at j. Improvement for $\boldsymbol{x}$ is

$$\begin{aligned} \Delta_{tail}(x, i, y, j) = &S_e^y(j) - S_e^x(i) \\ &- (\gamma_n^y - \gamma_{j-1}^y)(\delta_j^y + \delta_{i-1} + d_{i-1,idx(y,j)}). \end{aligned} \tag{15}$$

Similarly to *Exchange*, improvement of *Swap* is a sum of improvement of the particular routes. Improvement of the whole operation is therefore

$$\Delta_{swap\_tails}(x, i, y, j) = \Delta_{tail}(x, i, y, j) + \Delta_{tail}(y, j, x, i),$$

which can be again determined in constant time.

4. Results

Performance of the proposed approach has been evaluated for both mTDP and mGSP. The experiments for mTDP were run on a set of standard instances from TSPLIB (Reinelt, 1991) with sizes between 52 and 1002 and various numbers of vehicles. As there are no benchmark instances for mGSP, instances from TSPLIB were also used for which probabilities of vertices were generated randomly. To ensure repeatability of experiments, a generator from `random.org` was utilized for generating 10000 normally distributed random numbers between 1 and 10 and the string "2016-09-11" was set as a seed[2]. Random numbers were assigned to vertices respecting the order, i.e. i-th vertex of a TSPLIB instance has assigned i-th random number. The numbers were furthermore normalized to make the sum of probabilities of all vertices for an instance equal to one.

[2]We are ready to publish the generated sequence if the paper is accepted

Concerning the parameters of the method, we set $\kappa = 2$, which provides good variability as well as the quality of generated initial solutions. The size of the sequences of 2-opt moves in intensification is $\alpha = 20$, $n_{depth} = 5$ for first 4 depths of backtracking, and $n_{depth} = 1$ for the rest. N_{it} is set to 50, generating a total of 100 initial solutions. Starting positions of all vehicles are all set to the first vertex of the instance.

The proposed GRASP approach was compared with two existing methods. The first one is a variant of classical cluster-first, route-second approach. It splits the vertices into M clusters using k-means and then runs a GRASP meta-heuristic proposed in Kulich et al. (2016) for each cluster. More specifically, we employed k-means++ – an augmentation of k-means which significantly improves both the speed and the accuracy of k-means (Arthur and Vassilvitskii, 2007). The second approach is our algorithm (Kulich et al., 2017) which employs greedy heuristics for clustering instead of k-means. According to our knowledge, this algorithm provides currently-best solutions for both mTDP and mGSP. In the next text we refer to the methods as `kmeans` for the cluster-first, route-second approach employing k-means for clustering, `heuristic` for the approach presented in Kulich et al. (2017) and `proposed` for the proposed method.

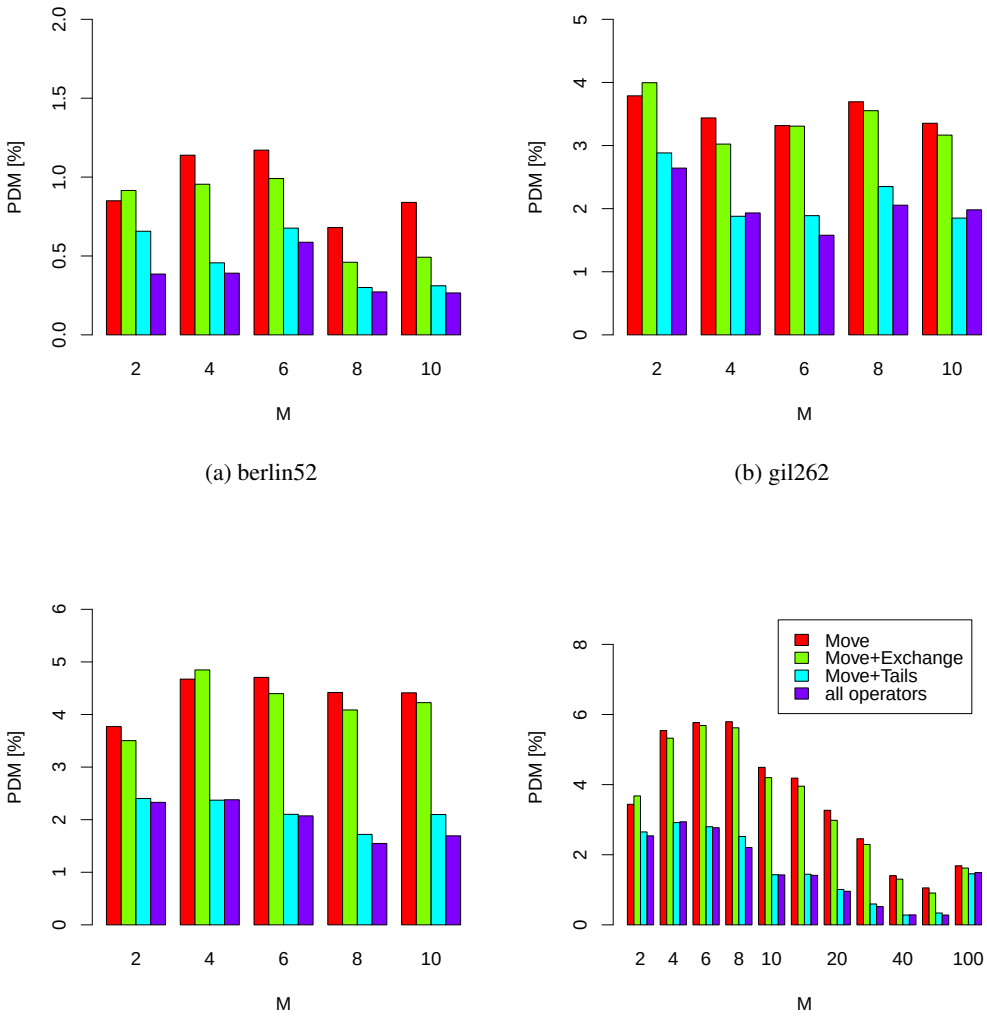
All experiments were performed within the same computational environment: a standard PC with an Intel®Core i7-3770 CPU at 3.4 Ghz running OS Sabayon with the Linux kernel 4.4.0. The algorithms have been implemented in C++ and compiled by clang 3.8.1. with "-O2" flag. 50 runs were run for each experimental setup consisting of an instance, the number of vehicles, and a method to provide statistically significant results.

The results for mTDP are presented in Table 1, where the meaning of the symbols is the following: M stands for the number of vehicles, BKS for the best-known solution (to the best of our knowledge, there is no other mTDP solver and BKS is thus the best solution found by one of the evaluated methods), SD is standard deviation, and T is execution time in ms averaged over all runs for a given setup. PDB is the percent deviation to BKS of the best solution values found by the algorithm (denoted as $best$), i.e. PDB=($best$-BKS)/BKS. Similarly, PDM is the percent deviation of the mean solution value to BKS. The best solution was found by the proposed approach for all instances, PDB for `proposed` is therefore omitted as it zero for all setups.

The results show that regarding quality, `proposed` outperforms the other methods in all cases. PDB's of `kmeans` and `heuristic` are worse on average by 19.5% and 8.2% respectively, while PDM's averaged over all runs are 1.3% for `proposed`, 23.8% for `kmeans` and 8.4 for `heuristic`. `proposed` provides better results by more than 33% than `kmeans` and by more than 15% in comparison to `heuristic` in extreme cases.

Regarding computational times, `proposed` is faster than the other methods for small problems and small numbers of vehicles, while `heuristic` is fastest for the other problems. This is most significant for *pr1002* problem containing 1002 vertices, where the proposed method is slower by three orders than the others.

Similarly, results for mGSP are summarized in Table 2 in the same way as for mTDP. Again, PDB for `proposed` is omitted as it is zero for all cases. All running times slightly increased in comparison to mTDP, but ratios remain similar. Also quality of generated solutions did not change much, `proposed` outperforms both `kmeans` and `heuristic` by 18.3% and 9% respectively on average regarding PDB, while average PDM's are 1.6% for `proposed`, 22.3% for `kmeans` and 9.3 for `heuristic`.



(a) berlin52

(b) gil262

(c) u724

(d) pr1002

Fig. 2: Influence of operators on solution quality

4.1. Influence of operators

We also studied the influence of operators working with two routes on quality of a generated solution and time complexity. A set of experiments was performed for mGSP instances with the same settings as in the previous case for four variants of the proposed GRASP scheme differing in operators used.

The *Move* operator only was used in the first variant, *Move* and *Exchange* were employed in the second variant, *Move* and *Swap Tails* in the third one, and all three operators in the last one.

The resulting graphs for the selected problems are shown in Figs. 2 and 3. It can be seen that all three operators have an influence on solution quality and that combination of all three operators leads to best results in the majority of cases. On the other hand, the difference to the combination of *Move* and *Swap Tails* is not big. Furthermore, this combination provides better solutions in the rest of cases.

Justification of usage of *Exchange* can be seen if Fig. 3 which shows computational time of the variants. The variant with *Exchange* thus reduces computational burden in all cases except *pr1002*, in many cases even significantly. This is especially the case of *u724*, where time is reduced twice in comparison to the combination of *Move* and *Swap Tails*.

4.2. Real world experiment

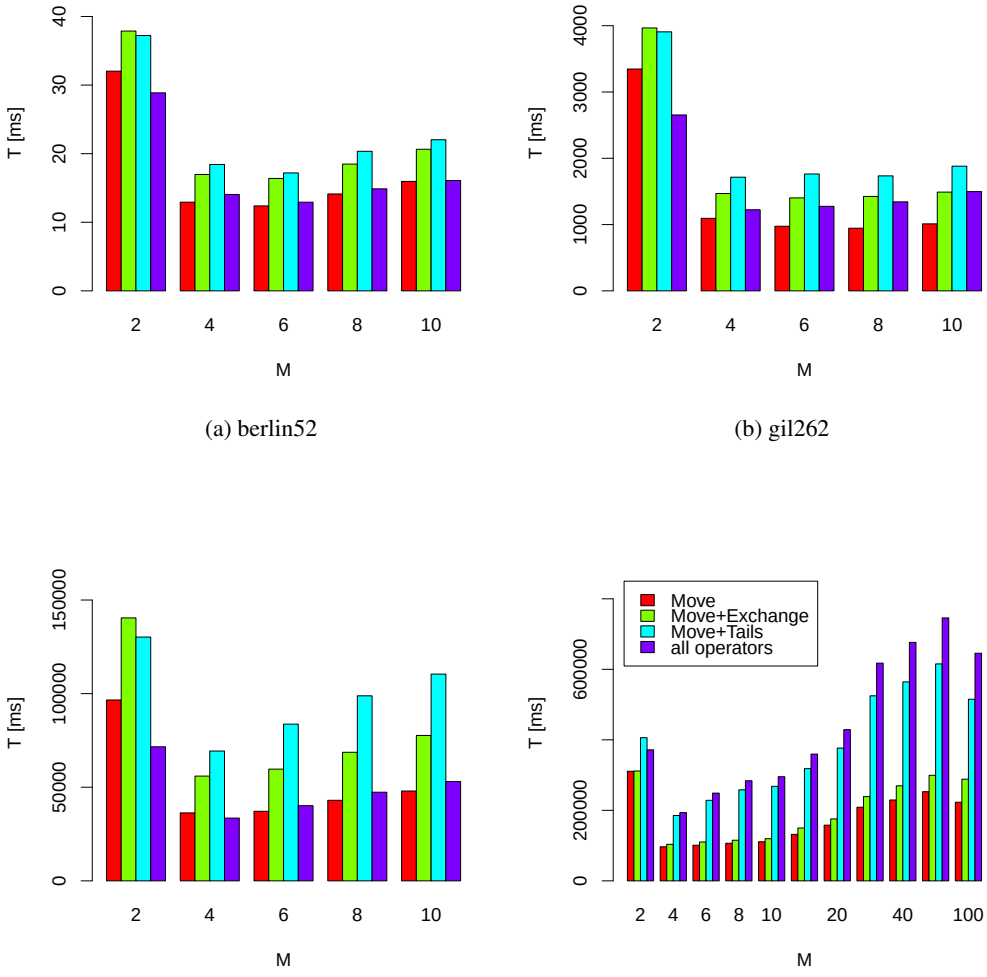
As a proof of concept that the proposed approach can be successfully deployed in real-world scenarios, we demonstrated it with real robots on the SyRoTek system (Kulich et al., 2013). SyRoTek is a platform for e-learning and distant experimentation in robotics and related areas consisting of thirteen robots equipped with standard robot sensors (laser range-finders, sonars, odometry, etc.) and it is freely available at `http://syrotek.felk.cvut.cz`. The robots operate in the Arena of size 3.5×3.8 m and are fully programmable and remotely controlled.

The test scenario was as follows. Assume a polygonal map of the environment in the form of a polygon with holes as depicted in Fig 4b and a team of two robots. The aim is to navigate the robots through the environment with the aim minimize expected time of finding an object of interest placed randomly in the environment. A robot can detect an object if their distance is smaller than a sensor range (0.3 m in our case) and there is no obstacle between the robot and the object.

In order to employ the proposed GRASP-based approach which minimizes Eq. 1, a graph $G(V, E)$ has to be built first. A conforming constrained Delaunay triangulation with constrains on a triangle area and its angles Shewchuk (2002) is used to generate a triangular mesh. Centers of generated triangles with a larger area than a predefined threshold form a set of vertices V, see Fig. 4b. We are aware that this is not the best solution as for example Kazazakis and Argyros (2002) or Faigl and Kulich (2006) produce less number of vertices, but the used approach is good enough to demonstrate the feasibility of the proposed planning algorithm.

A set of edges E is produced in two steps. First, two vertices are connected with an edge if they are mutually visible, i.e. there is no obstacle between them and a cost of the edges is the Euclidean distance of the vertices. All-pairs shortest path Johnson's algorithms Johnson (1977) is run to compute costs of edges between all pairs of vertices and thus to construct a full graph.

Weight of a vertex is computed as a sum of contributions of all points of the environment δ-visible from the given vertex (visible and closer to the vertex than a sensor range). Point contribution is determined as a $\frac{1}{K}$, where K is a number of vertices from which the point is δ-visible. In other words, each point in the environment has a contribution one which is equally divided to vertices from which it is δ-visible. The motivation for this computation is that a weight of a vertex corresponds to the area visible from it for the first time. As the order of vertices in which they will be visited is not known in advance, the probability that a point in the intersection of δ-visibility regions of K points will be first seen from a given vertex is



(a) berlin52

(b) gil262

(c) u724

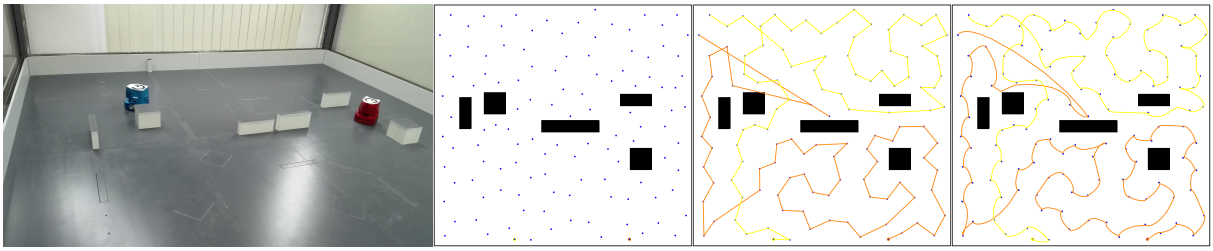
(d) pr1002

Fig. 3: Influence of operators on computational time

$\frac{1}{K}$.

Given $G(V, E)$, costs of edges, and weights of vertices, the proposed mGSP solver can be run to provide paths for particular robots. The robots are finally navigated along these paths running Smooth Nearness Diagram (SND) algorithm Durham and Bullo (2008) for robot motion control and collision avoidance. Robot's positions were taken from the localization system provided by SyRoTek.

Several experiments were done with the same setup. The robots during the mission are shown in Fig. 4a, the generated routes for one mission are depicted in Fig. 4c, while the trajectories traversed by robots are shown in Fig. 4d.



(a) (b) (c) (d)

Fig. 4: Real experiment. (a) The robots performing the mission. (b) The map with generated goals (note that obstacles were inflated by Minkowski sum enabling to plan non-colliding paths for circular-shaped robots). The starting positions are highlighted in yellow and orange. (c) Planned routes. (d) Trajectories traversed by robots.

5. Conclusion

The paper addresses the problem how to plan routes for a team of robots in order to find an object of interest placed randomly in a priori known environment. We formulated the problem as a multi-vehicle variant of the Graph Search Problem given a graph describing the environment and proposed a novel algorithm to this problem based on the Greedy Randomized Adaptive Search Procedure meta-heuristics which employs Variable Neighborhood Descent. The presented experimental results show that the method outperforms state-of-the-art approaches in quality of provided solutions for all tested setups. Moreover, computational time for instances containing less than 300 vertices is 3 seconds at maximum on a standard PC. These two properties qualify the approach to be employed in on-line decision making in robotics as a typical size of robotic problems rarely exceeds this size.

We also performed experiments for setups where weights of particular vertices are equal which leads to a solution of the multi-vehicle Traveling Deliveryman Problem as a particular case of mGSP. The results show that the proposed method is suitable for this problem too as ratios of solution quality and computation time to the compared approaches are similar to mGSP.

Future research will go in several directions. First, we would like to increase the scalability of the approach to allow solving problems with sizes over 1000 vertices in seconds. One of the possible streams is to integrate Tabu Search into GRASP. Another option how to reduce computational time is to explore only a subset of vertices for each operation instead of a full neighborhood. The intuition is that adjacent vertices in a route are generally close to each other in good solutions.

Another interesting stream would be to reformulate the problem to consider changes of vertices' weights according to the order the vertices are visited as indicated in Section 4.2. Incorporating of these

changes into the objective function and consequently to the algorithm should improve expected time of finding an object in realistic scenarios.

Finally, we would like to study properties of mGSP for multi-robot search in an unknown space similarly to using a mTSP solver for exploration task Faigl et al. (2012) and a GSP solver for search in a priori unknown environment Kulich et al. (2014, 2016).

Acknowledgement

This work has been supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 688117, by the Technology Agency of the Czech Republic under the project no. TE01020197 "Centre for Applied Cybernetics", the project Rob4Ind4.0 CZ.02.1.01/0.0/0.0/15_003/0000470, and the European Regional Development Fund.

References

Abeledo, H., Fukasawa, R., Pessoa, A., Uchoa, E., 2012. The time dependent traveling salesman problem: polyhedra and algorithm. *Mathematical Programming Computation* 5, 1, 27–55.

Afrati, F., Cosmadakis, S., Papadimitriou, C.H., Papageorgiou, G., Papakostantinou, N., 1986. The complexity of the travelling repairman problem. *Theoretical Informatics and Applications*

Arthur, D., Vassilvitskii, S., 2007. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 1027–1035.

Ausiello, G., Leonardi, S., Marchetti-Spaccamela, A., 2000. On salesmen, repairmen, spiders, and other traveling agents. In Bongiovanni, G., Petreschi, R. and Gambosi, G. (eds), *Algorithms and Complexity*, *Lecture Notes in Computer Science*. Vol. 1767. Springer Berlin Heidelberg, pp. 1–16.

Bektas, T., 2006. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega* 34, 3, 209–219.

Boone, N., Sathyan, A., Cohen, K., 2015. Enhanced approaches to solving the multiple traveling salesman problem. In *Proceedings of the AIAA Infotech@Aerospace Conference.*, American Institute of Aeronautics and Astronautics.

Dewilde, T., Cattrysse, D., Coene, S., Spieksma, F., Vansteenwegen, P., 2013. Heuristics for the traveling repairman problem with profits. *Computers & Operations Research* 40, 7, 1700–1707.

Durham, J.W., Bullo, F., 2008. Smooth nearness-diagram navigation. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pp. 690–695.

Faigl, J., Kulich, M., 2006. Sensing locations positioning for multi-robot inspection planning. In *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06)*, pp. 79–84.

Faigl, J., Kulich, M., Přeučil, L., 2012. Goal assignment using distance cost in multi-robot exploration. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ Int. Conf. on*, pp. 3741 –3746.

Feo, T.A., Resende, M.G., 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization* 6, 2, 109–133.

Fischetti, M., Laporte, G., Martello, S., 1993. The delivery man problem and cumulative matroids. *Operations Research* , March 2015, 1055–1064.

Geetha, S., Poonthalir, G., Vanathi, P., 2009. Improved k-means algorithm for capacitated clustering problem. *INFOCOMP Journal of Computer Science* 8, 4, 52–59.

Godinho, M.T., Gouveia, L., Pesneau, P., 2014. Natural and extended formulations for the Time-Dependent Traveling Salesman Problem. *Discrete Applied Mathematics* 164, 138–153.

Gouveia, L., Voß, S., 1995. A classification of formulations for the (time-dependent) traveling salesman problem. *European Journal of Operational Research* 2217, 93.

Hansen, P., Mladenović, N., 1997. Variable Neighborhood Search. *Computers & Operations Research* 24, 1, 1097–1100.

Heilporn, G., Cordeau, J.F., Laporte, G., 2010. The Delivery Man Problem with time windows. *Discrete Optimization* 7, 4, 269–282.

Johnson, D.B., 1977. Efficient algorithms for shortest paths in sparse networks. *J. ACM* 24, 1, 1–13.

Kazazakis, G.D., Argyros, A.A., 2002. Fast positioning of limited-visibility guards for the inspection of 2d workspaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002)*, IEEE, Lausanne, Switzerland, pp. 2843 – 2848.

Ke, L., Feng, Z., 2013. A two-phase metaheuristic for the cumulative capacitated vehicle routing problem. *Computers and Operations Research* 40, 2, 633–638.

Koutsoupias, E., Papadimitriou, C., Yannakakis, M., 1996. Searching a fixed graph. In Meyer, F. and Monien, B. (eds), *Automata, Languages and Programming*, *Lecture Notes in Computer Science*. Vol. 1099. Springer Berlin Heidelberg, pp. 280–289.

Kulich, M., Chudoba, J., Košnar, K., Krajník, T., Faigl, J., Přeučil, L., 2013. Syrotek – distance teaching of mobile robotics. *Education, IEEE Transactions on* 56, 1, 18–23.

Kulich, M., Miranda-Bront, J.J., Přeučil, L., 2016. A meta-heuristic based goal-selection strategy for mobile robot search in an unknown environment. *Computers & Operations Research* In press.

Kulich, M., Přeučil, L., Miranda-Bront, J.J., 2014. Single Robot Search for a Stationary Object in an Unknown Environment. In *2014 IEEE International Conference on Robotics and Automation*, pp. 5830–5835.

Kulich, M., Přeučil, L., Bront, J.J.M., 2017. On multi-robot search for a stationary object. In *2017 European Conference on Mobile Robots (ECMR)*, pp. 1–6.

Lin, S., Kernighan, B., 1973. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*

Luo, Z., Qin, H., Lim, A., 2014. Branch-and-price-and-cut for the multiple traveling repairman problem with distance constraints. *European Journal of Operational Research* 234, 1, 49–60.

Lysgaard, J., Wøhlk, S., 2014. A branch-and-cut-and-price algorithm for the cumulative capacitated vehicle routing problem. *European Journal of Operational Research* 236, 3, 800 – 810. Vehicle Routing and Distribution Logistics.

Mattos Ribeiro, G., Laporte, G., 2012. An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. *Computers and Operations Research* 39, 3, 728–735.

Méndez-Díaz, I., Zabala, P., Lucena, A., 2008. A new formulation for the traveling deliveryman problem. *Discrete Appl. Math.* 156, 17, 3223–3237.

Miranda-Bront, J.J., Méndez-Díaz, I., Zabala, P., 2010. An integer programming approach for the time-dependent TSP. *Electronic Notes in Discrete Mathematics* 36, 351–358.

Miranda-Bront, J.J., Méndez-Díaz, I., Zabala, P., 2013. Facets and valid inequalities for the time-dependent travelling salesman problem. *European Journal of Operational Research*

Mladenović, N., Urošević, D., Hanafi, S., 2012. Variable neighborhood search for the travelling deliveryman problem. *4OR* pp. 1–17.

Ngueveu, S.U., Prins, C., Calvo, R.W., 2010. An effective memetic algorithm for the cumulative capacitated vehicle routing problem. *Computers & Operations Research* 37, 11, 1877 – 1885. Metaheuristics for Logistics and Vehicle Routing.

Reinelt, G., 1991. Tsplib - a traveling salesman problem library. *INFORMS Journal on Computing* 3, 4, 376–384.

Ribeiro, C.C., Vianna, D.S., 2005. A grasp/vnd heuristic for the phylogeny problem using a new neighborhood structure. *International Transactions in Operational Research* 12, 3, 325–338.

Ribeiro, G.M., Laporte, G., Mauri, G.R., 2012. A comparison of three metaheuristics for the workover rig routing problem. *European Journal of Operational Research* 220, 1, 28–36.

Rivera, J.C., Murat Afsar, H., Prins, C., 2016. Mathematical formulations and exact algorithm for the multitrip cumulative capacitated single-vehicle routing problem. *European Journal of Operational Research* 249, 1, 93–104.

Salehipour, A., Sörensen, K., Goos, P., Bräysy, O., 2011. Efficient GRASP+VND and GRASP+VNS metaheuristics for the traveling repairman problem. *4OR* 9, 189–209.

Sarmiento, A., Murrieta-Cid, R., Hutchinson, S., 2004. A multi-robot strategy for rapidly searching a polygonal environment. In Lemaître, C., Reyes, C.A. and González, J.A. (eds), *Advances in Artificial Intelligence - IBERAMIA 2004, 9th Ibero-American Conference on AI, Puebla, México, November 22-26, 2004, Proceedings*, Springer, pp. 484–493.

Sathyan, A., Boone, N., Cohen, K., 2015. Comparison of approximate approaches to solving the travelling salesman problem and its application to uav swarming. *International Journal of Unmanned Systems Engineering* 3, 1, 1–16.

Shewchuk, J.R., 2002. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry* 22, 1–3, 21 – 74. 16th {ACM} Symposium on Computational Geometry.

Silva, M.M., Subramanian, A., Vidal, T., Ochi, L.S., 2012. A simple and effective metaheuristic for the Minimum Latency Problem. *European Journal of Operational Research* 221, 3, 513–520.

Problem	M	BKS	kmeans				heuristic				proposed		
			PDB	PDM	SD	T [ms]	PDB	PDM	SD	T [ms]	PDM	SD	T [ms]
berlin52	2	68713	9.18	18.90	6154	65	2.22	2.22	1	44	1.25	552	25
	4	36855	17.37	21.70	1317	33	7.84	7.84	0	20	0.48	211	12
	6	27994	22.40	28.77	940	19	9.18	9.18	0	9	0.38	123	13
	8	24263	19.21	28.33	1209	13	4.97	4.97	0	7	0.51	70	15
	10	22800	20.11	26.90	1197	9	4.91	4.91	0	5	0.54	77	15
bier127	2	2249045	4.79	15.44	168919	573	6.43	7.40	8825	393	3.63	34964	319
	4	1148181	6.70	27.07	81356	256	13.23	13.23	0	119	2.07	10534	122
	6	783613	29.98	37.48	37073	189	12.23	12.23	0	84	1.09	5056	104
	8	628096	28.77	39.89	31155	129	13.54	13.54	0	53	0.93	2187	115
	10	547501	28.36	40.17	22537	100	11.84	11.84	0	40	0.65	2338	130
gil262	2	149205	3.25	6.76	2561	4348	3.33	4.06	636	3209	1.51	1176	2188
	4	82251	19.33	19.89	183	2185	5.91	6.16	83	1060	1.38	499	995
	6	61598	21.15	25.98	2053	1198	6.22	6.27	28	634	1.55	385	1069
	8	52575	23.23	28.41	2067	841	5.19	5.21	7	470	1.35	285	1249
	10	48025	25.35	29.42	1529	615	4.72	4.72	3	346	0.81	201	1467
lin318	2	2967790	7.03	8.95	22516	6141	10.45	11.81	14860	6018	3.19	46446	3209
	4	1652372	11.81	17.49	43079	2288	9.54	9.86	2656	1720	1.72	13140	2013
	6	1265217	16.39	20.05	32703	1432	13.18	13.27	1011	970	1.40	7710	2453
	8	1076975	17.90	22.07	40888	1033	12.76	12.79	266	578	1.42	6162	2829
	10	975942	19.81	23.29	34525	760	15.72	15.73	220	582	1.57	5654	3334
pcb442	2	5273584	5.60	6.77	29620	24214	0.07	0.92	21018	14676	1.68	44107	13446
	4	2676970	21.66	22.06	9065	9347	6.45	6.92	7084	3871	1.41	17714	5038
	6	1872102	29.67	32.61	20854	4933	9.79	9.94	2247	1687	1.15	11618	4689
	8	1529430	33.02	36.71	37093	3108	5.09	5.21	1111	1294	0.84	4984	4552
	10	1344161	35.50	39.24	22193	2334	4.37	4.41	735	1015	0.67	3946	5217
rat575	2	959549	6.16	6.77	3390	70798	3.78	4.53	3022	39692	2.17	6389	30352
	4	496821	14.23	20.13	12550	25405	5.71	6.21	1100	10062	1.27	3204	12241
	6	345969	21.60	23.86	7821	11629	8.63	8.89	473	4373	1.41	1937	15137
	8	275558	27.76	29.31	4796	7482	9.94	10.07	186	2388	1.11	1443	19672
	10	234400	31.96	35.50	4146	5626	11.32	11.38	105	1464	1.32	1518	25918
u724	2	7214847	6.01	7.16	29926	80266	4.30	5.10	26656	60029	2.55	87912	65586
	4	3832465	11.44	15.03	81181	28066	1.91	2.40	7600	16984	1.74	25961	31949
	6	2725276	18.40	19.69	20709	15513	7.42	7.77	4646	8657	1.29	18374	38807
	8	2198999	23.62	25.68	23246	11430	10.32	10.46	1408	6532	1.22	13261	47640
	10	1899414	27.45	29.67	29270	8838	11.26	11.31	866	3859	1.08	9223	57133
pr1002	2	62531684	4.92	5.50	194529	300765	5.09	5.94	256320	193190	2.30	638581	223976
	4	33311971	9.82	10.58	224774	85212	11.55	12.09	79727	55299	2.31	293324	97320
	6	23897982	13.08	14.34	404059	45121	11.47	11.95	48038	27321	1.97	226279	121200
	8	19169589	18.63	20.34	476898	31752	9.27	9.52	24908	14990	1.62	134931	139253
	10	16494748	18.70	21.81	660805	23205	19.56	19.83	20578	9670	1.31	86057	148598
	15	13126541	23.50	28.10	260240	15023	11.72	11.79	6968	5880	1.14	63299	199791
	20	11753981	24.06	27.22	206195	10195	10.10	10.12	2341	3361	0.83	40410	251827
	30	10630016	25.11	27.63	159026	7064	9.59	9.62	2259	1974	0.35	18346	373446
	40	10203089	29.64	32.19	173794	7062	6.83	6.84	1091	1833	0.37	14307	426098
	50	10032496	31.25	33.78	108641	6802	5.78	5.78	637	1742	0.23	9057	478671
	100	9890230	33.26	35.55	102717	6910	0.92	0.92	22	968	0.10	3729	421952

Table 1: Comparison of the algorithms for mTDP instances. The number in a problem name specifies a number of vertices.

Problem	M	BKS	kmeans				heuristic				proposed		
			PDB	PDM	SD	T [ms]	PDB	PDM	SD	T [ms]	PDM	SD	T [ms]
berlin52	2	1183.28	8.97	18.23	96.7	67	11.74	11.74	0.1	58	0.39	6.9	29
	4	660.59	13.27	18.20	22.9	40	7.78	7.78	0.0	24	0.39	2.3	14
	6	526.19	16.84	20.37	11.9	22	8.20	8.20	0.0	12	0.59	1.8	13
	8	471.17	13.47	18.05	14.1	15	8.44	8.44	0.0	8	0.27	1.1	15
	10	447.11	12.47	17.64	15.3	11	3.49	3.49	0.0	4	0.27	1.1	16
bier127	2	16395.26	4.46	14.58	1231.8	638	3.20	3.94	60.6	472	3.37	236.0	311
	4	8356.24	16.76	30.32	589.4	319	7.89	7.99	7.6	176	2.60	101.2	131
	6	5806.67	27.36	37.72	449.9	238	10.89	10.89	0.3	114	2.00	43.0	116
	8	4716.36	25.26	37.43	278.9	153	12.32	12.32	0.0	64	1.09	30.9	121
	10	4148.67	28.50	37.17	206.0	124	8.89	8.89	0.0	42	1.32	24.8	135
gil262	2	533.07	4.09	7.68	12.3	4378	4.44	5.68	2.8	3543	2.64	6.1	2654
	4	302.30	17.59	18.21	0.9	2679	7.56	7.83	0.4	1037	1.93	2.4	1223
	6	229.23	19.89	24.07	7.9	1452	5.20	5.32	0.2	613	1.28	1.9	1275
	8	197.43	21.84	27.01	6.6	1042	5.36	5.40	0.1	383	2.05	1.1	1343
	10	183.43	21.93	26.69	5.8	811	6.78	6.78	0.0	318	1.06	1.0	1497
lin318	2	9038.73	6.35	9.16	70.6	6901	13.57	14.39	36.5	6598	3.09	126.6	3802
	4	5053.88	13.10	18.13	66.6	2549	11.17	11.59	8.6	1759	2.35	49.6	2370
	6	3890.79	16.19	20.61	94.8	1804	12.84	13.02	4.9	875	1.45	31.6	2676
	8	3325.37	18.17	23.07	131.6	1316	12.12	12.14	0.6	604	1.22	18.1	3062
	10	3049.70	18.38	22.47	127.9	1054	14.92	14.93	0.4	547	1.16	14.7	3421
pcb442	2	11066.99	5.44	8.65	166.6	19777	8.88	9.87	49.8	14081	2.83	107.8	15780
	4	5786.41	20.99	21.68	20.7	8756	5.11	5.44	11.2	3411	2.85	44.2	5815
	6	4181.07	24.76	29.00	86.7	5195	6.24	6.37	3.5	1743	1.38	27.4	5451
	8	3454.99	28.90	31.91	69.3	3694	5.19	5.25	1.2	1125	0.68	12.0	5118
	10	3048.55	32.15	34.49	43.0	2930	4.90	4.91	0.3	729	0.85	9.1	5617
rat575	2	1553.78	5.07	5.90	6.6	51279	4.73	5.95	8.5	30045	1.92	11.6	39367
	4	816.14	15.23	18.36	19.8	20830	8.16	9.10	2.8	8578	1.31	5.4	15545
	6	574.14	19.41	22.06	13.9	12318	12.88	13.16	0.9	4679	1.76	4.3	18645
	8	461.13	25.81	27.52	8.0	8815	14.47	14.66	0.4	2735	1.78	2.8	21831
	10	398.31	29.63	32.41	7.4	6901	11.48	11.60	0.3	1862	1.58	2.9	25953
u724	2	9459.41	6.03	7.23	39.3	88676	7.95	9.42	48.8	65825	2.33	92.6	71599
	4	5004.09	11.38	15.62	108.0	31175	9.34	10.31	17.4	17132	2.38	50.1	33528
	6	3598.73	18.85	20.15	33.8	19127	10.41	10.64	4.7	10722	1.98	26.5	40147
	8	2930.00	24.66	26.43	32.0	14217	15.14	15.36	2.6	6698	1.55	19.4	47348
	10	2547.98	27.90	30.08	40.0	11010	10.10	10.20	1.5	4547	1.69	15.8	53018
pr1002	2	58480.15	5.28	6.37	286.7	272616	3.16	4.34	281.8	243931	2.54	571.7	371600
	4	31385.39	10.02	10.86	109.3	78987	14.88	15.88	110.2	53754	2.94	343.4	193579
	6	22583.08	14.45	15.75	326.0	51723	10.89	11.29	52.5	33497	2.77	201.9	248863
	8	18382.21	17.96	20.61	449.5	36915	13.23	13.49	19.9	19485	2.21	152.6	283945
	10	16008.10	17.61	21.39	564.0	27326	10.94	11.07	12.4	11881	1.42	91.3	295323
	15	12865.68	20.78	26.09	322.2	17006	12.94	13.01	5.5	6989	1.41	59.3	359504
	20	11608.85	22.83	26.15	216.7	12541	10.47	10.50	2.9	4006	0.96	36.3	428881
	30	10546.04	24.09	26.01	190.1	8821	9.14	9.16	1.6	2113	0.52	18.4	617506
	40	10169.98	27.70	29.48	105.6	8459	7.72	7.72	0.2	1515	0.28	11.4	676410
	50	9995.37	30.46	31.82	133.3	8488	5.79	5.79	0.1	1333	0.28	7.6	745812
	100	9871.34	30.98	33.05	79.9	8343	0.90	0.90	0.0	1004	0.07	2.9	645638

Table 2: Comparison of the algorithms for mGSP instances.