Applied Intelligence (2022) 52:9574–9596
https://doi.org/10.1007/s10489-021-02941-y

Path planning algorithm ensuring accurate localization of radiation sources

David Woller[1,2] · **Miroslav Kulich**[1]

Accepted: 13 October 2021 / Published online: 7 January 2022

Abstract

An autonomous search for sources of gamma radiation in an outdoor environment is a domain suitable for the deployment of a heterogeneous robotic team, consisting of an Unmanned Aerial (UAV) and an Unmanned Ground (UGV) Vehicle. The UAV is convenient for fast mapping of the area and identifying regions of interest, whereas the UGV can perform highly accurate localization. It is assumed that the regions of interest are identified by the UAV during an initial reconnaissance, while performing a simple motion pattern. This paper proposes a path planning algorithm for the UGV, which guarantees accurate source localization in multiple preselected regions and minimizes the total path length. The problem is formulated as the Generalized Travelling Salesman Problem (GTSP) defined for discrete sets of suitable maneuvers (circular arcs), ensuring source localization in the given regions. The problem is successfully solved by a modified version of the state of the art GTSP solver, Generalized Large Neighborhood Search with Arcs ($GLNS_{arc}$). Apart from adapting the GLNS, other aspects of the planning task are addressed: problem discretization and informed sampling of valid circular arcs, variants of weighting the nonrestricted trajectory segments between the arcs and postprocessing of the discretely planned trajectory in the continuous domain.

Keywords Search for radiation sources · Combinatorial optimization · Generalized Large Neighborhood Search · Generalized Travelling Salesman Problem · Heuristics · Metaheuristics

1 Introduction

Localization of radiation sources is a critical task repeatedly arising in various accidents of high seriousness. This paper focuses on localizing isolated point sources in otherwise uncontaminated areas or hot spots in large-scale accidents. Several types of incidents happened over the last century and are likely to occur again, no matter the level of technological progress and safety precautions [36]. First, there are cases of lost, stolen, or orphaned sources, commonly from an industrial or medical application. Sources were found at junkyards, abandoned factories, or even urban areas, often by unsuspecting citizens [31]. Another danger is related to the military use of nuclear energy and weapons. The U.S. alone admits 32 so-called broken arrow incidents (e.g., accidental nuclear detonation, contamination, loss in transit, or accidental jettisoning), including six cases of lost and never recovered nuclear weapons [35]. Finally, a number of accidents happened in the nuclear power industry, most notably the Chernobyl and Fukushima disasters. The initial postdisaster cleanup at Chernobyl included liquidation of highly radioactive debris, representing another relevant application.

Robotic systems are an obvious choice for radiation source localization due to the extreme risk the radiation presents to humans. As the environment can be urban or rural and indoor or outdoor, the commonly used systems are typically semiautonomous or entirely teleoperated. Unmanned Aerial Vehicles (UAVs) or Unmanned Ground Vehicles (UGVs) are deployed depending on the application, as each of the platforms has its advantages. UGVs are typically easier to navigate indoors, can operate for a longer time, and are capable of more accurate localization. On the other hand, UAVs are significantly faster and usable in

✉ David Woller
wolledav@cvut.cz

Miroslav Kulich
kulich@cvut.cz

[1] Czech Institute of Informatics, Cybernetics and Robotics, Czech Technical University in Prague, Prague, Czech Republic

[2] Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Karlovo náměstí 13, Praha 2, 121 35, Czech Republic

impassable terrain. In some applications, combining both platforms is beneficial, as their advantages can be combined.

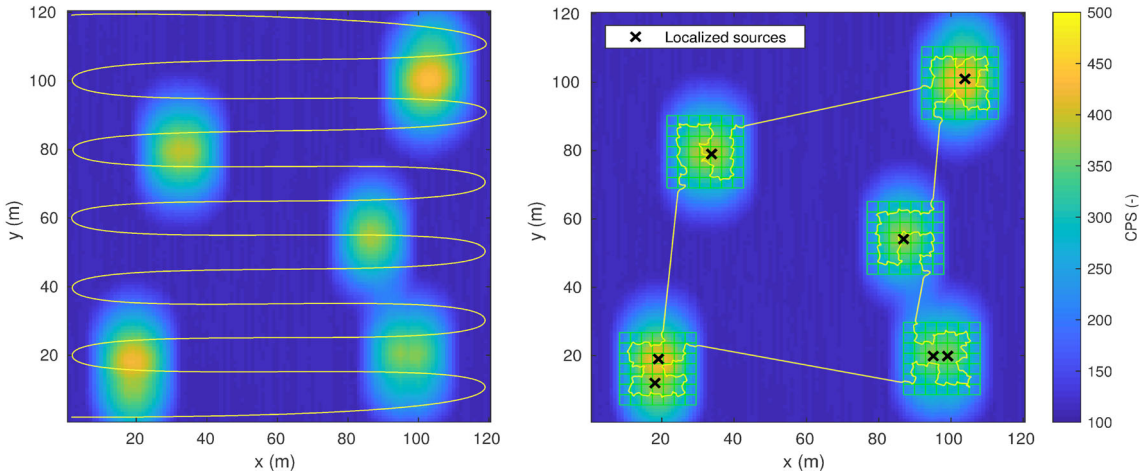
Such an approach to a semiautonomous search for sources of gamma radiation is presented in [11, 23] and [24]. The ultimate goal is to localize gamma radiation sources in an outdoor area, such as a place of a nuclear accident. The localization is to be carried out as quickly and precisely as possible, with subsequent use of an UAV and an UGV. The authors of [24] designed and constructed such a multirobot system, performed physical experiments, and evaluated the accuracy of the detection.

The considered scenario consists of two phases, which are illustrated in Fig. 1. First, the area is mapped by the UAV carrying a photogrammetry multisensor and a gamma detector. This phase's objective is to build a 3D map of the area surface and to pick regions with the potential presence of radiation sources. The UAV can carry only a lightweight gamma detector and its operation time is limited. Therefore, it is capable of detecting stronger radiation sources with insufficient precision (up to several meters). In case of weaker sources, the UAV might not be able to reliably distinguish the background radiation from source radiation, as documented in [11]. As there is no previous knowledge about the source position, the UAV performs an exhaustive search along a zig-zag trajectory while keeping an altitude of 10 meters above the terrain. The flight altitude was experimentally determined in [11] and it guarantees identifying all regions that could contain a source of activity relevant in the context of radiation protection. The minimal activity level of a potentially dangerous source was set to 10 MBq. The UAV trajectory and the discovered regions of interest are shown in Fig. 1a.

Second, the UGV with a more accurate gamma detector is deployed to inspect all previously discovered regions of interest. Thus, the scenario is not cooperative, as the UGV is deployed after the UAV finishes the initial mapping, not simultaneously. The UGV is substantially slower than the UAV, but it can operate for a longer time and locate the radiation sources more accurately. Multiple strategies such as simple zigzag pattern, Strong Source Search Algorithm [23], or Circular Algorithm [24] were proposed and tested. However, these algorithms are suitable only for single-source detection and do not utilize UAV-obtained information. This information was therefore processed by a human operator, who had to determine the regions of interest and manually define key segments (e.g., full circles of detection in the Circular Algorithm) of the UGV path. The success of this approach heavily depends on the operator experience. It does neither guarantee successful localization of all sources, nor does it optimize the UGV trajectory w.r.t. to any criteria. It is desirable to suppress the role of the operator and plan the UGV trajectory automatically and optimally w.r.t. some criteria, such as length or time.

This paper extends the preceding work by proposing a novel path planning algorithm for the UGV, which replaces the operator in the planning process and guarantees successful source detection. The goal is to plan an optimal UGV trajectory (e.g., with minimal length) or near-optimal one and to guarantee accurate localization of all radiation sources within the preselected regions of interest. An example of such a trajectory is shown in Fig. 1b, together with the detected positions of the radiation sources.

The UGV is assumed to be mounted with relatively inexpensive high-resolution gamma detectors, which measure only the count rate. A single radiation source's position can be reliably detected with such a setup by performing a specifically constrained maneuver in its close



(a) UAV trajectory and identified regions (b) UGV trajectory, searched subregions and detected sources

Fig. 1 Radiation intensity map produced by the UAV and illustration of the two-phase search

neighborhood. If this maneuver is a circular arc, and multiple detectors are mounted on the robot, a directional profile of the radiation can be constructed and used for accurate localization. Measuring along circular arcs is advantageous, as it results in steeper characteristics of the measured data. Thus, the peaks in the measured radiation intensity are more prominent and can be better distinguished from fluctuations in background radiation. Moreover, measurements along circular arcs can be easily interpolated, which is useful for determining the source position from the measured data. This setup was successfully deployed in [27, 34], or [23]. Task-specific conditions on the usability of the circular arc maneuvers are described in Section 2.2.4.

As the UAV-preselected regions of interest can be arbitrarily large and one circular arc may not be sufficient for covering a whole region, each region is divided into smaller subregions of fixed size (green squares in Fig. 1b). An unlimited number of valid maneuvers for exploring a subregion exists; therefore, infinitely many valid circular arcs can be sampled. Besides that, the order of subregion exploration is not fixed and is also subject to optimization. Therefore, after appropriate discretization of individual subregions, the planning problem can be reformulated as NP-hard Generalized Travelling Salesman Problem (GTSP). Given n nodes (circular arcs) divided into m sets (subregions), the goal is to find such a trajectory that passes through exactly one node from each set and is optimal with respect to some criterion, e.g., minimum length. The modified variant of the GTSP with circular arcs as vertices is from now on referred to as the GTSP_{arc}.

This paper presents an approach to the discretization of the planning problem, introduces GLNS_{arc} algorithm solving the discrete GTSP_{arc}, evaluates its performance in several experiments and proposes two improvements to the GLNS_{arc} functionality.

The contribution lies in multiple aspects.

- **Problem formulation.** The planning task is formulated as a discrete optimization problem, specifically a variant of the GTSP and called GTSP_{arc}. It is shown how to tailor the state of the art GLNS metaheuristic to solve it.
- **Automation of the UGV path planning.** The current solution relies on a human operator when processing the UAV-obtained information and specifying critical segments of the UGV trajectory. The algorithm presented in this paper replaces the operator role in this phase and enables to generate the UGV path automatically. The algorithm is meant to be run on a base station, after the initial UAV reconnaissance.
- **Informed discretization.** An infinite number of valid vertices can be sampled for each subregion, which increases the computation and memory demands. A method for detecting sampled vertices that are valid, but not useful in any potential solution is presented. Thus, the discretized problem size can be reduced considerably, which speeds up the planning and allows for covering much larger areas.
- **Guarantee of detection in regions of interest.** It is shown how to sample valid vertices that ensure source detection in predefined regions of interest. Currently, this is reliant on the operator experience.
- **Post-processing in the continuous domain.** The proposed GLNS_{arc} algorithm is capable of finding a locally optimal solution in the discrete domain. However, this solution may not be locally optimal in the continuous domain. Therefore, a local search procedure in the continuous domain is introduced. This procedure is deployed once in postprocessing, after the GLNS_{arc} finishes.
- **Optimality criteria.** The algorithm is capable of minimizing a custom optimality criterion, which can be difficult for the human operator. This criterion is determined by the weights assigned to edges in the GTSP_{arc} planning graph. Besides the standard weighting based on Euclidean distance, a more realistic weighting is introduced, which also considers the robot's rotation speed. It is also demonstrated that GLNS_{arc} can be deployed in an environment with obstacles or impassable terrain, both indoor and outdoor.

The paper is structured as follows. Section 1 contains this introduction and state of the art summary in Section 1.1. Section 2 provides the theoretical description of the problem and describes the exact steps of the solution. Sections 2.1 and 2.2 give the formal definition of the planning problem, as well as the specifics introduced by the application. Section 2.3 presents the employed GTSP solver - GLNS. Finally, Sections 2.4 to 2.7 tackle various aspects of the solution - most notably the necessary GLNS modifications, variants of edge weighting, an approach to the problem discretization and a local optimization technique in continuous space called DenseOpt. Section 3 then examines the behavior of the algorithm and evaluates the contribution of individual components of the solution. A concise summary and conclusion is given in Section 4.

1.1 State of the art

Concerning gamma radiation monitoring and search for radiation sources, there are various scenarios and therefore, various respective approaches. For the long term, static monitoring in places with a higher risk of radiation leak (e.g., a nuclear power plant, medical or industrial facilities), permanent sensor networks are typically installed, while the

current research is focused on developing reliable wireless sensor networks [7] or mobile sensor networks [26].

However, when the radiation source presence is not anticipated in the area or the area is too large or unsuitable for sensor network installment, a search mission deploying single or multiple measuring agents is a more adequate solution. Then, the motion planning becomes a critical factor in ensuring that the whole area is inspected and that the inspection is carried out in a reasonable time. The search process is often referred to as source seeking (SS) and determining the source position from the measured data as source term estimation (STE). Apart from radiation source localization, similar methods are also used for identifying leaks of gas or chemicals [25]. The SS motion planning methods can be classified either as adaptive or proactive [37]. In the case of adaptive methods, the motion planning is typically controlled by a feedback from the current output of STE, which is beneficial mainly for controlling the STE accuracy. Commonly used adaptive methods are gradient-based traversal, surging or casting [2]. These methods typically contain a simple mechanism for switching from one sensing location to another (towards another potential source), but this approach does not deal well with the intrinsic combinatorial problem, which is determining the order of exploration of individual regions. Therefore, they are suitable for localizing only a single source or for usage by a swarm of robots initially equally distributed across the area [8].

Proactive strategies are on the other hand designed to ensure full coverage of the area at the cost of higher time requirements and possibly lower accuracy. Several rather simple motion planning strategies, such as the zig-zag pattern [39] or contour mapping [13], were proposed for the outdoor environment. The main advantage of aerial spectrometry is the possibility of quickly exploring a relatively large area, while the main disadvantage is the low accuracy of source position estimation. For more accurate or indoor mapping, a ground-based agent such as a UGV can be employed. Compared to a UAV, the UGV typically moves at significantly lower speeds and has to deal with obstacles and impassable terrain. With no obstacles present or considered, some motion planning strategies used in aerial spectrometry were adapted to UGVs (e.g., the zig-zag pattern). However, these are not suitable in many real world applications and more complex motion planning strategies are needed. For example, [33] proposed an approach for an enclosed polygonal environment. This method first performs a convex polygon partitioning by splitting the environment into smaller regions explorable from a single position. Then, it determines the dwell time needed for staying in each of these positions using the Currie limit of detection [5] and plans a trajectory over all positions, ensuring exploration of the whole area. Similar approach was followed by [1], which formulated the multigoal mission as the Travelling Salesman Problem over a set of predefined measurement locations. The main difference to the $\mathrm{GTSP}_{\mathrm{arc}}$ is that the planning is performed over a set of predefined locations of static measurements, where the robot needs to stop. In the $\mathrm{GTSP}_{\mathrm{arc}}$, the robot takes useful measurements while constantly moving, so the $\mathrm{GLNS}_{\mathrm{arc}}$ plans over a set of maneuvers, rather than positions. In other words, the existing approaches directly solve the TSP or the GTSP over a fixed set of points while connecting these points with trajectory segments corresponding to a particular vehicle model, as in, e.g., the Dubins GTSP [19] are not directly applicable in $\mathrm{GTSP}_{\mathrm{arc}}$.

Another family of approaches, information sampling (IS) techniques [3, 17] extensively sample the robot configuration space to find a path/trajectory minimizing a given objective function. The number of samples and thus the computational complexity, however, grows exponentially with the number of dimensions. The configuration space of maneuvers in $\mathrm{GTSP}_{\mathrm{arc}}$ has six dimensions, in contrast to two or three dimensions in typical applications of IS. Therefore, even small $\mathrm{GTSP}_{\mathrm{arc}}$ instances will be hard to solve with current IS techniques. Moreover, the search in IS is restricted to a limited time horizon, while in the $\mathrm{GTSP}_{\mathrm{arc}}$ we plan a path that visits all regions.

When using both a UAV and a UGV in an outdoor environment, the advantages of both platforms can be combined. The UAV can be used for the fast yet inaccurate estimate of radiation source locations and the UGV for subsequent accurate localization. Therefore, there is no need for time-consuming exploration of the whole environment by the UGV. An example of such an application is presented in [4], where the area is first inspected by a UAV, which collects RGB images and radiation data of the area. Based on the collected data, a cost map for ground-based motion is created, and the UGV is then deployed to inspect regions of interest using repeatedly the A* algorithm (thus being suitable for detecting a single radiation source).

As the $\mathrm{GTSP}_{\mathrm{arc}}$ is an extension to the GTSP, and a GTSP solver was employed and modified in order to solve the $\mathrm{GTSP}_{\mathrm{arc}}$, this subsection mainly discusses state of the art in GTSP solvers. The GTSP is a combinatorial optimization problem extensively studied in operations research with many practical applications, such as location routing problems, material flow system design, post-box collection, stochastic vehicle routing or arc routing [21].

There are multiple methods of finding the optimal GTSP solution in exponential time. The GTSP can be modeled as an integer linear program (ILP) and solved with an ILP solver. One of the first formulations was introduced in [22]. Formulations studied by [10] then inspired the problem-specific exact branch-and-cut algorithm [9]. Another six different integer programming formulations are compared

in [30], with an emphasis on 'compact' formulations (i.e., formulations, where the number of constraints and variables is a polynomial function of the number of nodes in the GTSP). A commonly used approach is to transform the GTSP into the TSP using the Noon-Bean transformation [28]. Then, an exact TSP solver such as the Concorde [18] can be deployed.

Due to GTSP NP-hardness, new approaches on how to find acceptable solutions to large problem instances in polynomial time are still being proposed. The problem can be transformed into the TSP and solved with a heuristic-based TSP solver, e.g., the LKH [14], but numerous heuristic approaches were adapted specifically for the GTSP. Among these, the following three algorithms can be considered as the most successful. The first one is the GLKH solver [16], which is based on the Lin-Kernighan k-opt heuristic used in the LKH. The GLKH is reported to be tested on large-scale instances with up to 17180 sets and 85900 vertices, and it is focused on finding high-quality solutions. The second one is a memetic GK heuristic proposed in [12], which combines genetic algorithms with a local search procedure. The GK yields high-quality solutions with excellent runtime on medium-size GTSP instances, but it does not scale well for problems with more than 200 sets [6]. Finally, Smith and Imeson presented an algorithm combining adaptive large neighborhood search, simulated annealing, and two local search procedures. The algorithm is called Generalized Large Neighborhood Search (GLNS), and it is documented to often outperform both GLKH and GK on several GTSP libraries [32]. It dominates the other two solvers, especially on highly constrained nonmetric instances, whereas the GLKH performs best on the largest clustered Euclidean instances and the GK on medium size metric instances. The proposed approach to GTSP_{arc} is built on GLNS due to its most consistent performance over a broad portfolio of GTSP instances compared to the other two solvers.

2 Methods

2.1 Planning task formulation

This subsection provides a formal definition of the GTSP and elaborates on the complications arising from differences between the practical task of searching for radiation sources and the following standard GTSP definition.

Problem 1 (The Generalized Travelling Salesman Problem) Assume a complete weighted graph $G = (V, E, w)$ and a partition of V into m sets $P_V = \{V_1, ..., V_m\}$, where

- V is a set of n vertices
- $V_i \cap V_j = \emptyset$ for all $i \neq j$
- $\bigcup_{i=1}^{m} V_i = V$
- E is a set of edges such that all vertices are connected, apart from vertices from the same set
- w is a mapping assigning a weight to each edge $w : E \rightarrow \mathbb{R}$

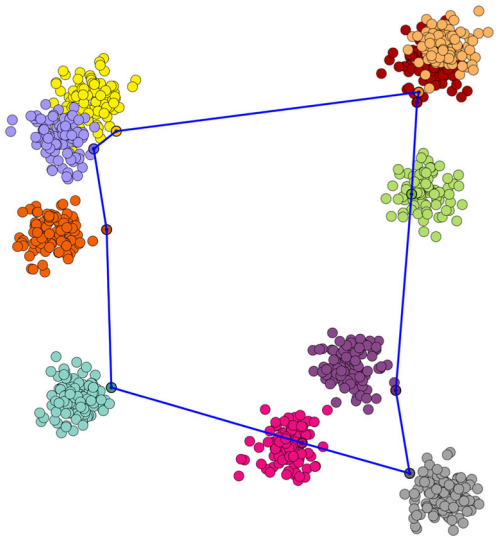


Fig. 2 Solved GTSP instance 10C1k.0 from MOM-lib [15]

Lets also define a tour T over graph G as a closed sequence of vertices and edges $T = (v_0, e_0, .., v_{m-1}, e_{m-1})$, where each edge connects two consecutive vertices - $e_i = (v_i, v_{i+1})$ and $e_{m-1} = (v_{m-1}, v_0)$. A set of vertices present in the tour T is denoted as V_T, a set of edges as E_T.

Then, the objective is to find a tour in G that contains exactly one vertex from each set and has a minimum length, i.e., it minimizes the tour length $w(T)$ defined as

$$w(T) = \sum_{e \in E_T} w(e).$$

An example of a solved GTSP instance is shown in Fig. 2, where vertices of the same color belong to the same set.

2.2 Application specifics

There are aspects of the planning task solved that prevent us from directly using the previously given GTSP formulation and already implemented solvers. Due to the following, the formulation has to be slightly changed, and the solver is appropriately modified.

2.2.1 Vertex definition

Contrary to GTSP, where a vertex is typically a point in 2D, a single vertex in the GTSP_{arc} represents a circular arc -

Table 1 Vertex parameters in the $GTSP_{arc}$

Symbol	Parameter description
x, y	planar coordinates of circular arc center (m, m)
r	arc radius (m)
α	angle between coordinate frame x-axis and arc axis (rad)
ω	angular size of the arc (rad)

a special trajectory segment such that passing by it allows a precise radiation source detection in a corresponding subregion. A minimal vertex representation consists of the parameters given in Table 1, which are also depicted in Fig. 3a.

2.2.2 Vertex weighting

As each vertex represents a segment of a robot trajectory, its length influences the total trajectory cost. In the standard GTSP definition given in Section 2.1 only edges, not vertices, have weights assigned. Therefore, this difference must be taken into consideration while implementing various metrics in the algorithm. In $GLNS_{arc}$, the weight w of each vertex is defined as the length of the arc: $w = \omega r$.

2.2.3 Vertex connecting

There is no restriction on the direction in which a vertex is to be passed, therefore connecting two vertices is ambiguous. The original algorithm considers only the possibility that edge weights depend on the vertex order - an edge from a to b might have a different weight than an edge from b to a (asymmetric GTSP). However, in the $GTSP_{arc}$, even with a fixed order, there are still four ways to connect two consecutive vertices, as shown in Fig. 3b.

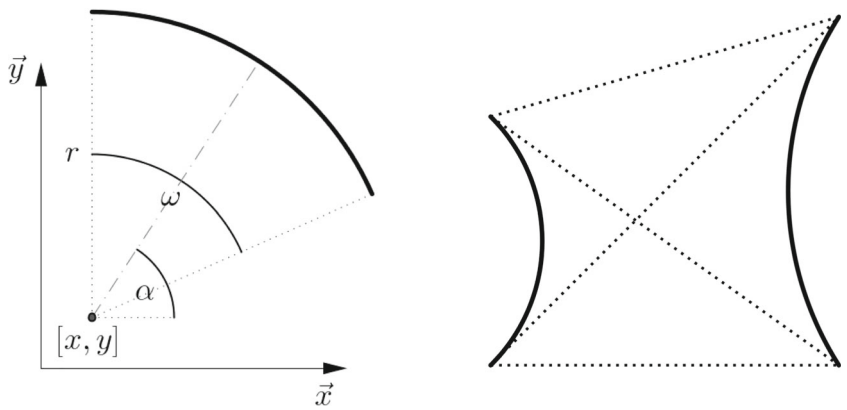
2.2.4 Vertex validity constraint

While detecting sources of radiation along circular arcs, only some arcs are potentially useful. Here, the vertices are generated to fit the experimental setup presented in [24], which uses a UGV fitted with a GPS unit, two mutually shielded gamma detectors, and counting electronics. The detection system is capable of measuring the rate of gamma radiation in counts per second (CPS) and recording the position of the measurement. It is partially directional due to the use of two detectors - when moving along a curve, it can be determined on which side of the curve the radiation source is located.

The STE procedure used for determining source locations based on the measurements taken is called the Circular Algorithm; it is described in [23] in detail and works as follows. The theoretical shape of the radiation intensity I_m measured in CPS along a circle near a radiation source can be described (according to the inverse square law and the law of cosines) as

$$I_m(\phi) = \frac{I_s}{a^2 + r^2 - 2dr\cos\phi}, \tag{1}$$

where I_s states the source intensity in CPS in a distance of 1 meter from the source, a is the distance from the source to the circle's center, r is the radius of the circle, ϕ is the difference between the angular coordinates of the UGV and the source with respect to the circle center and d is the distance from the source to the closest circle point. The function given in (1) can be used to interpolate measurements taken along a circular segment, although according to [23], quadratic interpolation is sufficient for determining the location of the intensity peak and thus the direction towards the radiation source. As it is also known, whether the source lies inside or outside of the circular segment (or better, the corresponding full circle) and which subregion is covered by this particular segment (see Section 2.6.1), an initial estimate of the source position



(a) Parameters definition (b) Vertex connecting

Fig. 3 Vertices (arcs) in the $GTSP_{arc}$

Table 2 Vertex validity constraint - parameters

Symbol	Parameter description
r	radius of circular arc (m)
d	distance from the source to the closest segment point (m)
l	arc length (m)
θ	angle between arc axis and line from arc center to source (rad)
I_s	intensity of the radiation source in CPS (-)
I_B	background radiation in CPS (-)
K	experimentally set constant (-)

and intensity can be made. This estimate is then iteratively improved by applying the Gauss-Newton method, which is reported to achieve an average accuracy of 17.8 cm RMS (root mean square) in real-world experiments carried out by the authors of [23].

It must be distinguished whether the peak in radiation intensity measured along a circular segment corresponds to a radiation source or fluctuations in the radiation background. Given a source $s = [x_s, y_s]$ with an intensity I_s, a robot position $p = [x_p, y_p]$ and a level of background radiation I_B, the intensity I_m measured by the robot is equal to

$$I_m = \frac{I_s}{(x_s - x_p)^2 + (y_s - y_p)^2} + I_B$$

according to the inverse square law. Then, the ratio between the minimal and maximal intensity I_m measured along a circular segment must be higher than a constant K called prominence [24] to identify a radiation source. For a circular segment, this constraint can be rewritten as

$$\frac{\frac{I_s}{2r(r+d)\left(1-\cos\left(\frac{l}{2r}-|\theta|\right)\right)+d^2} + I_B}{\frac{I_s}{d^2} + I_B} < K. \quad (2)$$

Individual parameters are described in Table 2 and their numerical values used for vertex generation are given in Table 3 (if constant). The values of I_s and I_m are in practice estimated from the initial UAV measurements. In Fig. 1, I_s corresponds to the highest CPS values at the yellow hot spots and I_B to the lowest readings in the dark blue areas. The measured intensity I_m is taken along the yellow line, marking the robot trajectory in Fig. 1b.

Table 3 Parameter values used for V_{val} generation

Parameter	Values
$x(m)$	-5 to 5 sampled by 1
$y(m)$	-5 to 5 sampled by 1
$r(m)$	1 to 5 sampled by 1
$\alpha(rad)$	$\pi/6$ to 2π sampled by $\pi/6$
$\omega(rad)$	$\pi/6$ to π sampled by $\pi/6$
$I_s(-)$	6000
$I_B(-)$	150
$K(-)$	0.7

2.3 Generalized Large Neighborhood Search (GLNS) description

GLNS is a GTSP solver introduced by S. L. Smith and F. Imeson [32]. This subsection gives only a brief overview of its functioning. A full description of the algorithm, including a thorough performance comparison with other approaches, can be found in [32].

Algorithm 1 GLNS.

Data: A GTSP instance (G, P_V)
Result: GTSP tour T
for $i \leftarrow 1$ **to** `cold_restarts` **do**
  $T \leftarrow$`initial_tour`(G, P_V)
  $T_{best,i} \leftarrow T$
  Initialize the acceptance temperature τ
  repeat
    Select a removal heuristic R and an insertion heuristic I
    Select number of vertices to remove N_r
    $T_{new} \leftarrow T$
    Remove N_r vertices from T_{new} using R
    Insert N_r vertices to T_{new} using I, one from each set not visited by T_{new}
    Locally re-optimize T_{new} (MoveOpt, ReOpt)
    if $w(T_{new}) < w(T_{best,i})$ **then**
      $T_{best,i} \leftarrow T_{new}$
    end
    if `accept`(T_{new}, T, τ) **then**
      $T \leftarrow T_{new}$
      Record improvement made by R and I
    end
  until stop criterion met
  Update selection weights of heuristics
  Update the acceptance temperature τ
end
return tour $T_{best,i}$ that attains $\min_i w(T_{best,i})$

GLNS implements an adaptive large neighborhood search, which is a metaheuristic planning approach based on the iterative application of constructive and destructive procedures to a current solution. These procedures are being selected randomly, using the roulette wheel selection mechanism. The selection weights of individual procedures are dynamically adjusted throughout the search process, according to their success in previous iterations. Each time a procedure improves the currently best solution, its

selection weight after the next restart is increased and vice versa.

GLNS pseudocode is given in Algorithm 1. First, an initial random tour is generated (line 2). Then, the following process runs iteratively. A pair of removal and insertion heuristics is selected according to their selection weights (line 6). These heuristics are then applied to remove N_r vertices from the current tour T and insert different N_r vertices, thus creating a modified tour T_{new} (lines 7 to 10). The modified tour T_{new} is subject to the local optimization techniques MoveOpt and ReOpt (line 11) and is consequently accepted or declined, while using a simulated annealing criterion (line 15). This process repeats until one of the stop criteria is met (line 19). After that, the planner updates the selection weights of the heuristics (line 20) and either starts the whole process again with a new cold restart or returns the best tour found overall.

The acceptance criterion (line 15) uses a simulated annealing procedure, which allows for accepting nonimproving tours T_{new} with a small probability depending on the temperature τ. The temperature τ is initialized at the beginning of each cold restart (line 4) and then gradually decreases in so-called initial descent. Then, after a certain number of nonimproving iterations, the temperature is increased again. This is called a warm restart, which also ends after a fixed number of nonimproving iterations. The temperature updates are performed at the end of each iteration (line 21). Each cold restart then consists of an initial descent and several warm restarts.

2.4 Proposed GLNS modifications towards GLNS$_{\text{arc}}$

As described in Section 2.2, graph vertex in the GTSP definition corresponds to a circular arc in the GTSP$_{\text{arc}}$. This arc represents a part of a trajectory and has certain properties, that have to be taken into account while employing GLNS to solve the GTSP$_{\text{arc}}$. The main issues arise from the fact, that the arc has a nonzero length and that connecting two vertices is ambiguous. Necessary modifications to the algorithm solving these issues are described in detail in this subsection. To prevent confusion, the word vertex is used when talking about circular arcs (i.e., the graph vertices in GTSP$_{\text{arc}}$) from now on.

2.4.1 Vertex duplication

In Section 2.2, a vertex is described by this tuple of parameters - $\langle x, y, r, \alpha, \omega \rangle$. This representation is sufficient for problem formulation but impractical for implementation, as it requires distinguishing between various ways of vertex connecting. Instead of doing that, an additional parameter $sign \in \{\pm 1\}$ is added. This parameter determines in which direction the vertex is to be passed through (-1 for clockwise passage, +1 for anticlockwise). Naturally, this doubles the total number of vertices in GLNS$_{\text{arc}}$ algorithm, as each vertex is inserted with both possible $sign$ values. On the other hand, the problem with edge connecting is solved, because the original GLNS allows for solving asymmetric GTSP, which is the case now.

2.4.2 Tour weight

Let $T = (v_0, e_0, v_1, e_1, ..., v_{m-1}, e_{m-1})$ be a tour and $w(T)$ its weight. In GLNS$_{\text{arc}}$, this weight is calculated as

$$w(T) = \sum_{i=0}^{m-1} w(e_i) + \mathbf{\sum_{j=0}^{m-1} w(v_j)}, \tag{3}$$

where v_i, e_j are the vertices and edges from T and $w(v)$, $w(e)$ are their respective weights. The bold expression in (3) and in all the following equations is newly added in GLNS$_{\text{arc}}$, and it reflects the fact that vertices have nonzero weight.

2.4.3 Cheapest insertion and unified insertions

GLNS contains several insertion heuristics that add vertices from currently unused sets to an incomplete tour T according to some simple rules. In all of these heuristics, the insertion cost of a vertex $v_{new} \in V_i$, $V_i \in P_V \setminus P_T$ is minimized. Here, P_T contains those sets, that are already visited by T. In the cheapest insertion, this cost is minimized to select both V_i and v_{new}, whereas, in the remaining three unified insertions, V_i is already selected by a different mechanism and only v_{new} is sought. In all cases, the insertion cost c_{ins} has to be modified to take the weight of v_{new} into account:

$$c_{ins} = w(v_j, v_{new}) + w(v_{new}, v_{j+1}) - w(e_j) + \mathbf{w(v_{new})}.$$

Here, v_j and v_{j+1} are two consecutive vertices in T before insertion and $w(v_j, v_{new})$, $w(v_{new}, v_{j+1})$ and $w(e_j)$ weights of corresponding edges.

2.4.4 Worst removal

Similarly to insertion heuristics, GLNS contains several removal heuristics, removing vertices from a tour T while using some simple rules. Worst removal removes such vertex $v_j \in V_T$ from tour T, that maximizes the removal cost c_{rem}. GLNS$_{\text{arc}}$ modification is again rather straightforward:

$$c_{rem} = w(e_{j-1}) + w(e_j) - w(v_{j-1}, v_{j+1}) + \mathbf{w(v_j)}.$$

2.4.5 ReOpt

Re-Opt, which is a local optimization subroutine, attempts to optimize the choice of vertices while keeping the set order fixed. This is achieved by performing a graph search through all sets, in which only edges between two consecutive sets are considered. When expanding from vertex $x \in V_i$ to vertex $y \in V_{i+1}$, current score in y is calculated as

$$score(y) = score(x) + w(x, y) + w(y).$$

Moreover, the first vertex $a \in V_1$ is initialized with $score(a) = w(a)$, instead of zero.

2.4.6 MoveOpt

MoveOpt subroutine attempts to optimize the set order by randomly removing a vertex v_i from a tour T and reinserting another vertex v_j from the same set to any position in the tour so that the insertion cost is minimized. This cost c_{ins} is modified the same way as in the cheapest and unified insertions, i.e.

$$c_{ins} = w(v_j, v_{new}) + w(v_{new}, v_{j+1}) - w(e_j) + w(v_{new}).$$

2.4.7 Remarks

Some parts of GLNS were not formally modified, although the original idea behind them might have changed in GLNS_{arc} due to task reformulation.

Set-vertex distance from a set V to a vertex u is still defined as

$$dist(V, u) = \min_{v_i \in V}(\min(w(u, v_i), w(v_i, u))).$$

In GLNS_{arc}, these distances are precomputed after vertex duplication. Therefore, the value obtained corresponds to the shortest path to or from u to V, no matter the $sign$ of u, v (= their orientation) or the edge (u, v) direction.

A different situation arises in the distance removal heuristic. The motivation is to remove vertices from a current tour T, which are "close to each other". At each iteration, a vertex v_{seed} is selected randomly from the set of already removed vertices V_{rem}. The next vertex v_j to be removed from T is obtained as

$$v_j = \arg\min_{v_j \in T}(\min(w(v_{seed}, v_j), w(v_j, v_{seed}))).$$

Here, GLNS_{arc} considers only edges between vertices v_{seed}, v_j and not their oppositely oriented variants available in the GTSP_{arc} instance, as these variants are not present in T.

2.5 Edge weighting variants

Two variants of edge weighting in the GTSP_{arc} graph were designed in the GLNS_{arc}. The first one ($line$) calculates the edge weight as the Euclidean distance between the connected vertices endpoint and entry point. This metric is commonly used in GTSP and similar problems, but it may be of limited value in GTSP_{arc}, as it does not reflect the rotation time needed at the vertex endpoints. Two edges with a similar weight may thus result in considerably different trajectory execution times. The second variant ($lineWA$) is designed to reflect this, as it also considers the rotation time needed at each endpoint.

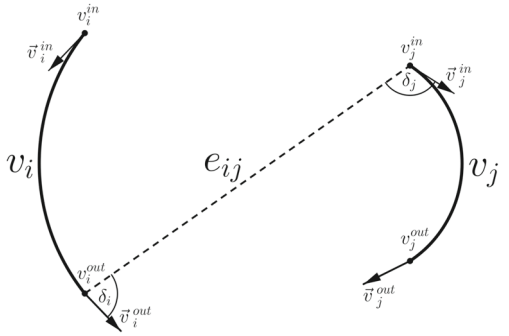


Fig. 4 Edge parameters

Let v be a vertex in GTSP_{arc}, with parameters as in Table 1. The vertex is oriented according to $sign$. Therefore, it has an entry point v^{in} and a leaving point v^{out}. The robot orientation in these points is then given by the tangent vector $\mathbf{v}^{in}$, respectively $\mathbf{v}^{out}$, as shown in Fig. 4. Definitions of individual edge weighting variants follow.

2.5.1 line

Let us consider vertices v_i, v_j connected by an edge e_{ij} (from v_i to v_j). The edge weight is then calculated as

$$w(e_{ij}) = \|v_i^{out} - v_j^{in}\|$$

2.5.2 lineWA - line with weighted angles

Similarly to the $line$ type, this variant connects vertex endpoints with a straight line. Let us again consider vertices v_i, v_j and an edge e_{ij}. Then, let δ_i be the smaller angle between vector $\mathbf{v}_i^{out}$ and e_{ij} (and δ_j between $\mathbf{v}_j^{in}$ and e_{ij} respectively), as shown in Fig. 4. The edge weight is then defined as

$$w(e_{ij}) = \|v_i^{out} - v_j^{in}\| + k(\delta_i + \delta_j), \tag{4}$$

where k is an application specific constant, reflecting the robot in-place turning speed.

2.6 Problem discretization

As was explained in Section 1, a single set of vertices in the GTSP_{arc} corresponds to a subregion with the potential

presence of a radiation source. A vertex in the GTSP_{arc} then corresponds to a maneuver along a specific circular arc. Equation (2) restricts the parameters of the circular arc, so that accurate localization of the radiation source is guaranteed. This constraint needs the source position to be known; therefore, it cannot be directly applied when sampling valid arcs covering the whole subregion - precise source position is not known yet. Thus, the following straightforward approach was adapted.

2.6.1 Sampling sets - covering subregions with vertices

The procedure of sampling valid vertices in GTSP_{arc} is described in Algorithm 2.

Algorithm 2 Sampling of valid vertices.

```
   Data: Subregion R, ranges of x, y, r, α and ω
   Result: Set of valid vertices V_val
1  V_val ← ∅
2  Uniformly sample p sources s within R
3  for x ∈ range(x) do
4      for y ∈ range(y) do
5          for r ∈ range(r) do
6              for α ∈ range(α) do
7                  for ω ∈ range(ω) do
8                      Generate vertex
                       v ← v(x, y, r, α, ω)
9                      valid ← true
10                     for i ← 1 to p do
11                         if ¬ constraint(s_i, v) then
12                             valid ← false
13                             break
14                         end
15                     end
16                     if valid then
17                         V_val ← V_val ∪ {v_sign=1}
18                         V_val ← V_val ∪ {v_sign=−1}
19                     end
20                 end
21             end
22         end
23     end
24 end
25 return V_val
```

The algorithm takes the following inputs: finite sets of possible vertex parameter values x, y, r, α, and ω (described in Table 1) and a subregion R. First, the subregion R is uniformly densely covered with p potential source positions s (line 2). Second, a circular arc is generated for each combination of given parameter values (line 8). If the newly generated vertex satisfies (2) for all p source positions, the arc is assumed to be valid for the whole subregion (line 9 to 13). The newly generated valid vertex v is then added to the set of valid vertices V_{val} in both orientations given by $sign$ (lines 16 to 18).

2.6.2 Reducing set size

The sampling procedure described in Section 2.6.1 generates a set of valid vertices V_{val}, whose size grows rapidly, proportionally to the range of input parameters x, y, r, α and ω. However, some vertices sampled may not be actually useful. A procedure for utilization-based V_{val} reduction is proposed in this section.

The contribution of some vertex v_i from a set V_{val} to the total tour weight $w(T)$ depends only on its neighbors v_{i-1} and v_{i+1} in a tour T. If some vertex $v_i \in V_{val}$ does not minimize the partial cost

$$c_{par} = w(v_{i-1}, v_i) + w(v_i) + w(v_i, v_{i+1}) \tag{5}$$

for any possible configuration of its neighbors v_{i-1} and v_{i+1}, it is redundant and can be removed from V_{val}.

To perform the vertex utilization analysis, a square grid of points is generated around the set V_{val}. An example of such a grid applicable for edge type $line$ is shown in Fig. 5a. Points in this grid correspond to the endpoints of vertices v_{i-1}, v_{i+1} and they are sufficient for determining the relevant edge weights $w(v_{i-1}, v_i)$, $w(v_i, v_{i+1})$. A sufficient set of potentially usable vertices V_{suf} is then extracted from V_{val} by testing all possible combinations of these endpoints and keeping only those vertices from V_{val} minimizing (5) for any of these combinations. This process is described in Algorithm 3.

Algorithm 3 Utilization-based set reduction.

```
   Data: Original set V_val, grid parameters P
   Result: Reduced set V_suf
1  V_suf = ∅
2  Generate grid G = G(P) around V_val
3  for p_1, p_2 ∈ G × G do
4      c_best ← ∞
5      v_best ← null
6      for v ∈ V_val do
7          c_par ← w(p_1, v) + w(v) + w(v, p_2)
8          if c_par < c_best then
9              c_best ← c_par
10             v_best ← v
11         end
12         V_suf ← V_suf ∪ v_best
13     end
14 end
15 return V_suf
```

The accuracy of this method depends on the set of grid parameters P. For the edge type $line$, where edge weight

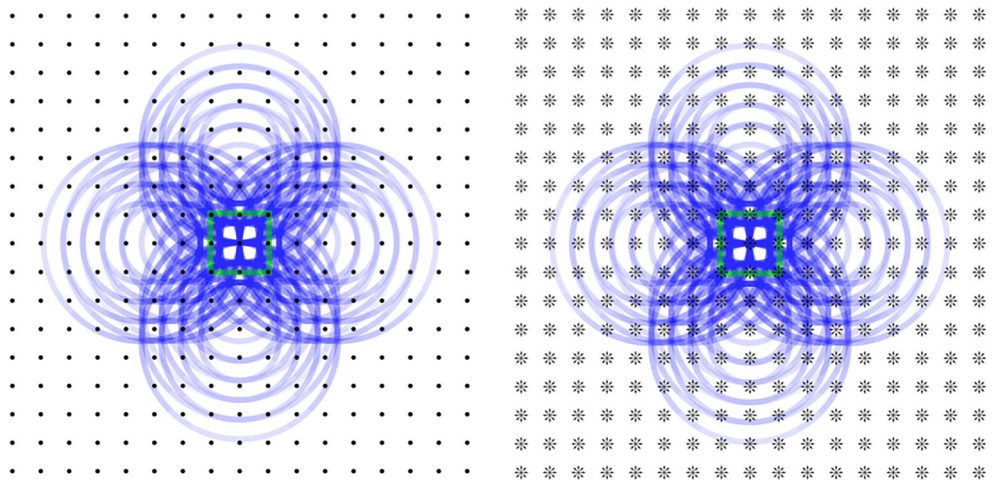


Fig. 5 Grids used for reducing V_{val} size

(a) Edge type *line* (b) Edge type $lineWA$

is calculated as the Cartesian distance between vertices endpoints, these parameters are grid size $size$ and grid resolution $posRes$. A point in the grid then corresponds to a point in 2D. In the case of the $lineWA$ type, which also considers the amount of rotation needed in each endpoint, a point in the grid consists of a point in 2D and a vector, as shown in Fig. 5b. Thus, there must be an additional parameter $angleRes$, which determines the angular resolution used when sampling multiple vectors in different directions from each point.

Suitable values of these parameters depend on the set V_{val} and on the edge type used. A simple method for extracting a sufficient set V_{suf} while refining the parameters P until the size of V_{suf} converges is proposed and described in Algorithm 4. First, V_{suf} is extracted using the reduction procedure described in Algorithm 3 with initial parameters P (line 1). Each parameter from P is then repeatedly refined according to the refinement step from Table 4 (line 5) and the V_{suf} is extracted again (line 6). Refining of the current parameter stops when there is no increase in the size of V_{suf} (line 8).

As the parameters are tuned sequentially, it may happen that refining the latter parameter enables further refining and subsequent increase in V_{suf} size for some of the preceding parameters. To cover this possibility, Algorithm 4 can be run repeatedly with the previously found parameters used as an initial solution in the next run, until there is no change in V_{suf} size over one whole run.

Table 4 Parameter values used for V_{suf} generation

Parameter	Initial value	Refinement step
$size(m)$	10	$size = size + 1$
$posRes(m)$	1	$posRes = posRes/2$
$angleRes(rad)$	$\pi/6$	$angleRes = angleRes/2$

Algorithm 4 Set reduction with parameters refining.

Data: Initial parameters P, original set V_{val}
Result: Tuned parameters P, reduced set V_{suf}

```
1  V_suf ← reduce(V_val, P)
2  for i ← 1 to |P| do
3      repeat
4          size = |V_suf|
5          Refine value of P[i]
6          V_suf ← reduce(V_val, P)
7          newSize = |V_suf|
8      until size = newSize
9  end
10 return P, V_suf
```

The approach presented does not guarantee that the generated set V_{suf} fully substitutes the original set V_{val} - especially when the refinement steps or the initial values of P are poorly chosen. However, the experiments presented in Section 3.4 show that there is no significant decrease in the quality of the solution obtained when using V_{suf} instead of V_{val} in GTSP$_{\text{arc}}$ instances, whereas the runtime is reduced dramatically and much larger areas can be covered.

2.7 DenseOpt optimization

As explained in Section 1, the planning task is originally continuous, as each set can be covered by infinitely many vertices, given that they respect the constraint in (2). Sampling a finite number of vertices is enforced by using GLNS, which is designed for discrete problems. The inevitable consequence is that the solution obtained by GLNS$_{\text{arc}}$ will probably never be optimal in the original continuous domain. This drawback is accepted, as GLNS$_{\text{arc}}$

is a metaheuristic-based planner, which does not give any guarantees about solution optimality as well. However, the quality of the final solution is negatively influenced by the sampling density. Apart from that, once the GLNS_{arc} reaches a good quality solution, further convergence tends to be slower and slower. In this phase, it may be more beneficial to perform a simple solution refining in the continuous domain, rather than spending more time solving the optimization problem in the discrete problem. This is the exact purpose of the proposed DenseOpt procedure.

DenseOpt is a newly proposed intensification optimization technique which performs local search in the continuous domain, after the solution in the discrete domain is returned by GLNS_{arc}. It is named to match the other two local optimization techniques defined in GLNS - MoveOpt, and ReOpt. Given a tour T obtained by GLNS_{arc}, DenseOpt searches the close neighborhood of each vertex in T and randomly samples new admissible vertices, not present in the discrete GTSP_{arc} formulation $G = (V, E, w)$. If the newly sampled vertex improves the weight of T, it replaces the originally present vertex from the same set. The process is described in Algorithm 5.

Algorithm 5 DenseOpt.

```
Data: Tour T = (v_0, e_0, v_1, e_1, ..., v_{m-1}, e_{m-1})
Result: Locally improved tour T
1  indices ← [0, 1, ..., m − 1]
2  repeat
3      stop ← True
4      Uniformly randomly shuffle indices
5      for j ← 0 to m − 1 do
6          index ← indices[j]
7          v ← v_index ∈ T
8          for k ← 1 to N_s do
9              Sample valid vertex v_new close to v
10             if v_new improves w(T) then
11                 stop ← False
12                 Replace v_index in T by v_new
13                 Update edges e_{index−1}, e_index in T
14             end
15         end
16     end
17 until stop = True
18 return T
```

The whole tour T is repeatedly optimized until there is no improvement in its weight. In each iteration, a random order of resampling is created by shuffling the array $indices$ (line 4). Then, each vertex in the tour T is resampled N_s times (lines 8-9) as v_{new}. Sampling of v_{new} is limited to a predefined range of parameters r, α, and ω. The original density of sampling in the GTSP_{arc} instance solved determines this range. Vertex v_{new} is sampled anywhere between its closest neighbors. If v_{new} improves tour cost $w(T)$, it is added to T and v_{index} is removed (lines 10-12). Moreover, the edges $e_{index-1}$ and e_{index} are newly generated, so that the newly added v_{new} is connected to the rest of the tour T (line 13). As the vertex v_{index} in T is being replaced continuously, its original position is stored in copy v, so that new vertices v_{new} are sampled in the same subregion (line 8). Parameter N_s was experimentally tuned (see Section 3.5) and set to $N_s = 300$.

3 Results

This chapter documents and interprets the experiments carried out. It is focused on thoroughly demonstrating GLNS_{arc} capabilities, performance, and assessing the contribution of the additional solution components. It does not provide a comparison with other methods, which is due to several reasons. First, the problem formulation is new and was not fully addressed before. Second, the application previously relied on a human operator input, which is not a feasible approach for thorough experimental comparison. Third, the relevant methods for similar problems are not directly applicable in the considered application without nontrivial adaptation.

The proposed algorithm considers three criteria:

1. source localization in a preselected region is to be guaranteed,
2. all preselected regions are to be inspected,
3. the total trajectory length over all regions is to be optimized.

Adaptive methods (such as [2, 23] or [24]) fully respect only the first criterion and can be iteratively applied to satisfy the second, given that the already discovered sources are removed before continuing the search. Proactive methods (e.g., the zig-zag pattern [39]) are designed to satisfy only the first criterion and are suitable for localizing multiple sources without extraction. However, no adaptation of a proactive method that would consider also the second and third criteria is known to us, apart from the proposed one. In the preceding work [23], the UGV trajectory was created manually and meeting all three criteria depended on a human operator experience. For obvious reasons, this approach is not suitable for experimental comparison.

All instances solved in fast and default mode were solved 50 times so that the results could be processed statistically. This number was reduced to 5 in case of the slow mode due to its excessive time requirements. All experiments were carried out on a single core of Lenovo P330 desktop PC with an Intel Core i7-8700, 3.2 GHz CPU, and 32 GiB of RAM. The GLNS_{arc} was implemented in C++.

3.1 Generating problem instances

It was described in Section 2.6.1, how to sample a set of valid vertices V_{val}, covering a subregion R. In all generated problem instances, the subregion R was set to a square with the dimensions of 3×3 meters. The size of the subregion is limited by the application-dependent parameters I_s, I_B, and K. Actual values used for generating V_{val} are given in Table 3, as well as values of constants used in the `constraint` function. In real-world experiments, the values of I_s and I_B are estimated from the UAV collected data.

The obtained set V_{val} is shown in Fig. 6a, where the subregion R is marked by the green line, and the generated vertices are displayed in blue. Vertices are partially transparent, so the darker shade of some segments indicates that multiple vertices are overlapping. In total, the set contains 3824 vertices.

The set V_{val} was subsequently reduced by performing the set reduction described in Section 2.6.2. An example of a reduced set V_{suf} is shown in Fig. 6b. This particular set is generated for edge type $line$ by reducing the set V_{val} from Fig. 6a. It contains only 120 vertices (compared to 3824 vertices in V_{val}). Initial values and refining steps for the parameters P are given in Table 4. Parameters common for $line$ and $lineWA$ edge types have identical values. Final values of grid parameters P are $size = 16$ meters and $posRes = 0.125$ meters.

If a region of interest discovered by the UAV in the initial phase is larger than 3×3 meters, or if there are multiple regions, they need to be split into smaller subregions of an appropriate size. E.g., a region of size 18×18 meters can be covered by 36 subregions of size 3×3 meters, as shown in Fig. 7, so that the successful radiation source detection is guaranteed in any point of the area. Each subregion corresponds to a set of vertices in the GTSP_{arc}, and vertices from the same set are displayed in the same color.

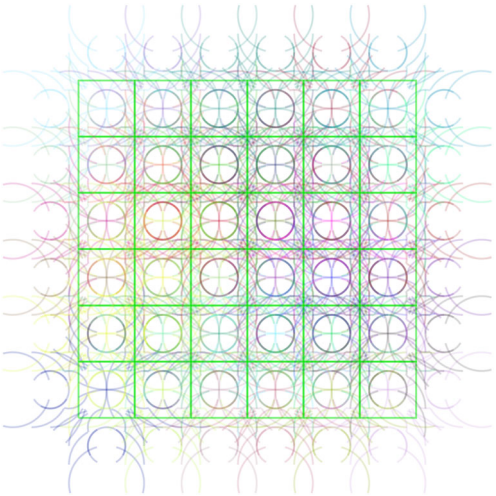
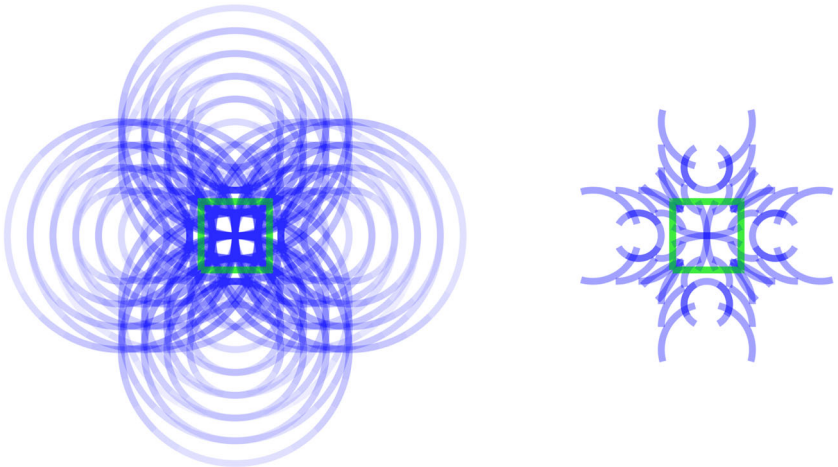


Fig. 7 Problem instance `tiles_suf/6x6_4320`

3.2 Datasets description

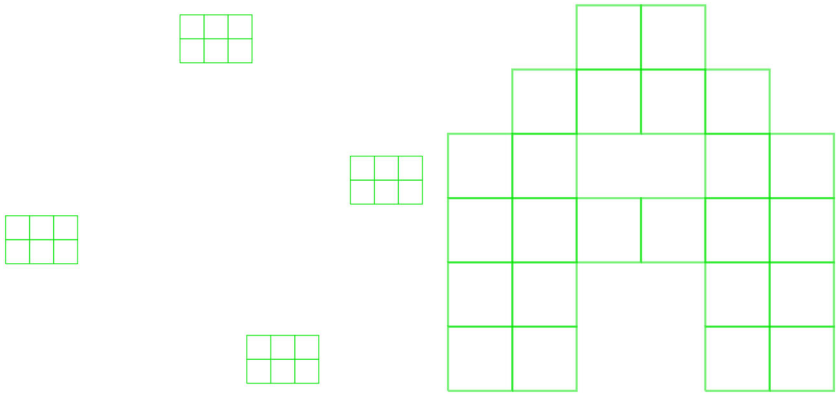
Three datasets were created to evaluate the GLNS_{arc} performance, behaviour and tune the denseOpt parameter: `tiles_full`, `tiles_suf` and `patterns`. The datasets are available at [38].

Dataset `tiles_full` consists of 14 problems of increasing size, which were created by placing the full set of 3824 valid vertices V_{val} described in Section 2.6.2 in a tiled pattern, similarly to the problem displayed in Fig. 7. The largest problem in this dataset covers an area of 12×15 meters, which corresponds to 20 sets (each covering 3×3 meters) and 76480 vertices.



(a) Full set V_{val} (b) Reduced set V_{suf}

Fig. 6 Generated sets of valid vertices

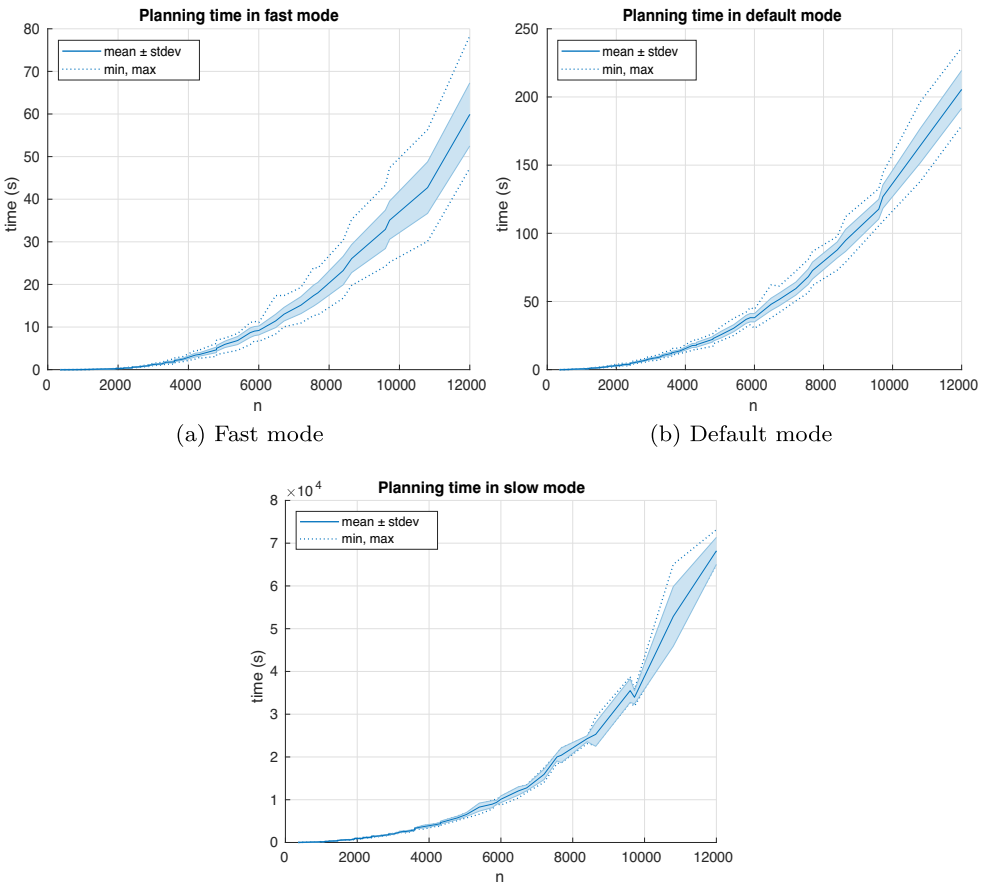


(a) clust6_range=40_v3 (b) bold_a

Fig. 8 patterns dataset - examples of instances

Dataset tiles_suf was created in the same manner, but it is based on the reduced set V_{suf} containing only 120 vertices, which was generated for edge type $line$. The dataset contains a total number of 53 problems, while the largest problem covers an area of 30 × 30 meters, corresponding to 100 sets and 12000 vertices.

Finally, dataset patterns is also based on the V_{suf} set, and it contains problems of medium size ranging between 24 and 32 sets. These problems are created with the intention to capture the behavior of the algorithm on structurally varied problems. It contains fifteen problems, where the sets are clustered by 3, 4, or 6, six problems with



(a) Fast mode (b) Default mode

(c) Slow mode

Fig. 9 Planning time across planner modes

sets arranged in a letter-shaped pattern, and ten problems with sets distributed randomly. Two examples are shown in Fig. 8.

3.3 GLNS$_{arc}$ modes of operation

GLNS comes with three sets of parameters corresponding to the following planner modes - fast, medium (default), and slow. These modes differ most notably in the number of cold and warm restarts, parameters determining the total number of iterations, and the frequency of applying the local optimization techniques MoveOpt and ReOpt. A full list of all values can be found in [32]. Individual settings were not further modified or tuned, but they were compared in terms of quality of solution and runtime on the `tiles_suf` dataset. The problems in this dataset have a very similar structure, but they gradually increase in size; thus they are suitable for scalability and applicability assessment of individual modes.

Figure 9 shows the planning time needed by individual modes in relation to the problem size, respectively, the number of vertices n. Consistently with the runtime analysis provided in [32], all dependencies plotted are polynomial.

Instances of up to 2700 vertices were always solved within 1 second in the fast mode; thus the GLNS$_{arc}$ can be used as an online planner for smaller problems. Mean planning time for the largest problem of 12000 vertices is 59.9 seconds, while in the worst case, the planning took 78.3 seconds. Fast mode time demands are therefore moderate even for larger instances. The best and worst-case planning times are circa 30% from the mean value, while the standard deviation is up to 10% of the mean. Planning times in the default mode (Fig. 9b) are about one order higher than in the fast mode and show slightly lower deviations (with extrema within 15% from the mean and standard deviation up to 6% of the mean). As for the slow mode (Fig. 9c), planning times are about three orders higher than in the fast mode, thus solving problems with more than 4000 vertices in terms of hours. Individual problems were solved at most 5 times due to the excessive time demands of the slow mode; therefore, the remaining statistical properties are not conclusive and comparable to the other modes.

As for the final tour weight obtained - GLNS$_{arc}$ performance in the fast mode on the `tiles_suf` is visualized in Fig. 10. The resulting weights show very low diversity - the worst tour weight is always within 2.5% from the best weight found and within 1.2% from the mean.

In the case of the default and slow mode, the diversity is even lower and not visually apparent when plotted in the same manner.

Figure 11 compares the relative difference of the mean best weight found across all three modes. Weights obtained in the slow mode are generally the best; thus they are being taken as a baseline. The difference is then calculated as $100\frac{w-w_{slow}}{w_{slow}}$, where w_{slow} is the mean final weight for the particular problem in the slow mode and w is the same value for the currently compared mode. Slow mode results are not displayed in the plot, as they would be compared to themselves, and the corresponding markers would naturally all lie on the line marking zero relative difference. The graph shows that the default mode is at most by 1.5% worse than the slow mode and the fast mode at most by 2.5%. The interpolated mean weight difference initially increases with problem size but appears to stabilize or even slightly decrease for the largest problem instances.

In conclusion, both the fast and default GLNS$_{arc}$ mode performance are sufficient for the intended application, as the planning requires several minutes at worst. The planning

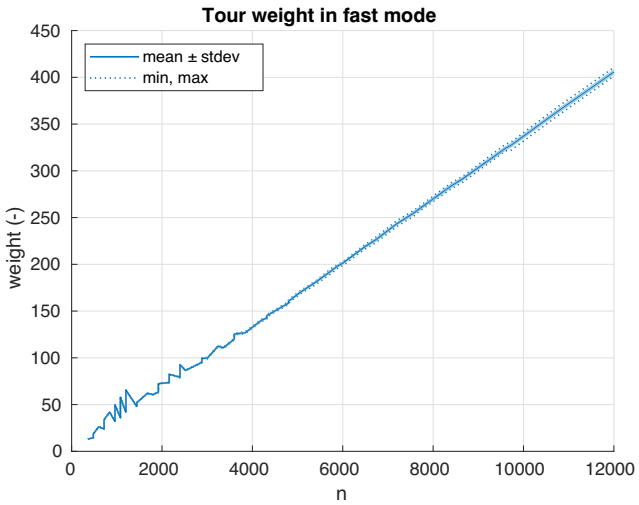


Fig. 10 Final tour weight in fast mode

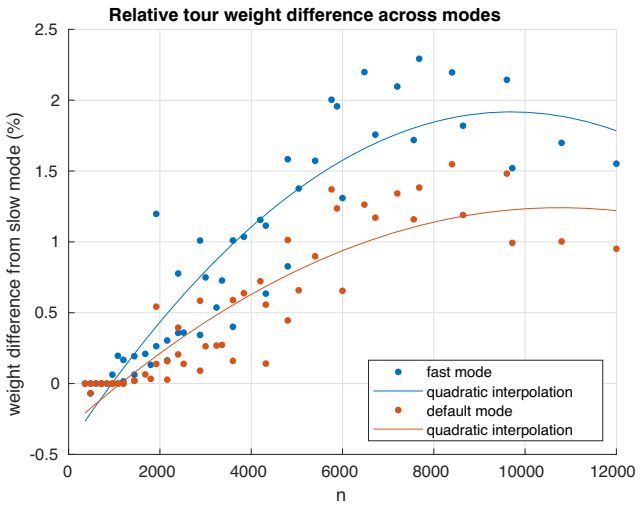


Fig. 11 Final tour weight comparison across planner modes

is to be carried out on a base station between the UAV and the UGV operation, so the plan is not needed instantly and minutes are acceptable for the operating staff. On the other hand, using the slow mode can be considered infeasible, as the planning times exceed hours. In terms of solution quality, the default mode produces solutions worse by up to 1.5% than the slow mode, but this gap can be easily closed by the proposed DenseOpt postprocessing technique, as documented in Section 3.5.

3.4 Planning with full vs. reduced sets

This subsection documents and evaluates the impact of reducing the set size (described in Section 2.6.2) on the planning time and the quality of the solution. For this purpose, the datasets `tiles_suf` and `tiles_full` are used. All problems from the `tiles_full` are present in the `tiles_suf`, meaning that the same area is being covered. However, the sampling density in the `tiles_suf` problems is much lower, as each subregion of 3×3 meters is covered by the reduced V_{suf} set (120 vertices), rather than by the full and naively sampled V_{val} set (3824 vertices).

Figure 12 shows the planning time needed for solving all problems in both datasets in the fast mode. The planning time is plotted against the number of sets m, as the number of vertices n differs greatly for the same problems. First, even though the planning times for the largest problems in the `tiles_full` dataset are not unacceptable (mean planning time is at most 167 seconds), the corresponding graph ends at $m = 20$, as the dataset does not contain larger problems. This is due to the fact that the limiting factor of the GLNS_{arc} are the memory requirements of storing all edge weights, not the planning time (at least in the fast mode). The current implementation could handle instances with circa $8-9 \times 10^4$ vertices on the hardware used. Second, the planning time needed for solving the `tiles_full` problems is significantly higher than for the `tiles_suf`. According to [32], GLNS time complexity in the fast mode in $O(mn)$. The number of vertices n is a linear function of m in both datasets, so the time complexity is $O(m^2)$, and the planning times should differ by a constant factor c, which can be estimated to $c = 300$.

Figure 13 shows the difference in the final tour weight for those problems in both datasets that are identical in terms of the number of sets and their distribution in space. Its value is calculated as $100\frac{w_{suf}-w_{full}}{w_{full}}$, where w_{full} is the mean final weight for a problem from `tiles_full` and w_{suf} is the mean final weight for the corresponding problem from `tiles_suf`. There is no apparent trend, but it can be said that the mean final tour weight of a problem from `tiles_suf` is at most by 0.1% worse than its counterpart from `tiles_full` and that this happens only for three problems. In the case of the remaining instances, the score obtained with V_{suf} is equal to or better than the score obtained with V_{val}. When compared to the relative differences across different planner modes shown in Fig. 11, the effects of using V_{suf} instead of V_{val} have a negligible impact on the quality of solution and the set size reduction described in 2.6.2 can be considered highly beneficial. Given that problems with $n = 75000$ are solvable, the GLNS_{arc} can be applied to problems with 625 V_{suf} sets, thus cover an area of 75×75 meters (5625 m^2). Without the reduction, the GLNS_{arc} could store at most 20 V_{val} sets covering an area of modest 180 m^2. Naturally, the size of the area depends on the parameters I_s and I_B, which are estimated during the initial UAV reconnaissance.

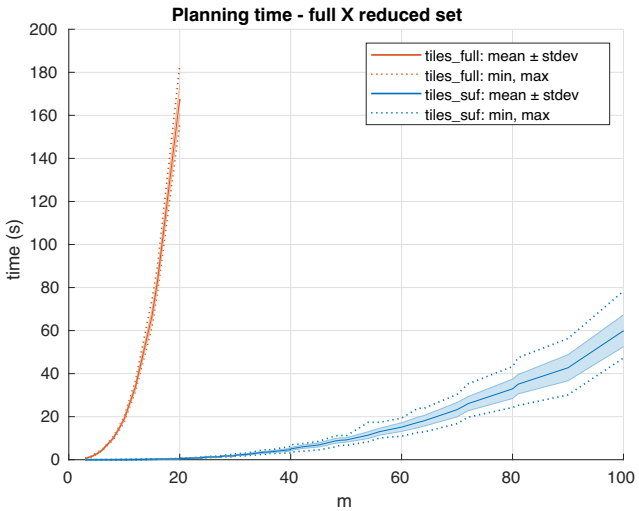


Fig. 12 Planning time comparison - full vs. reduced set

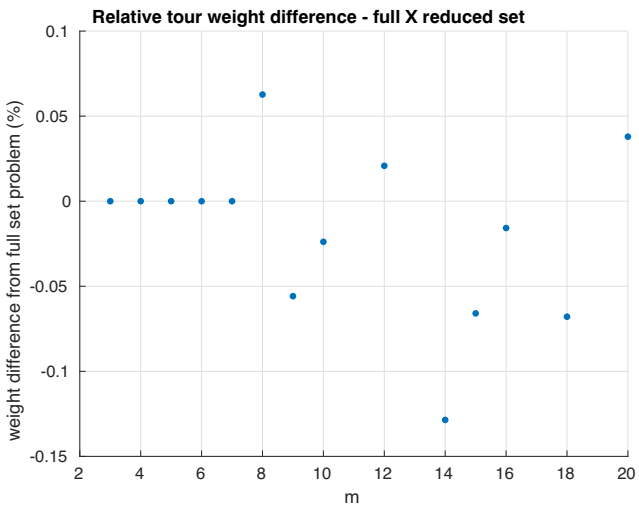


Fig. 13 Final tour weight comparison - full vs. reduced set

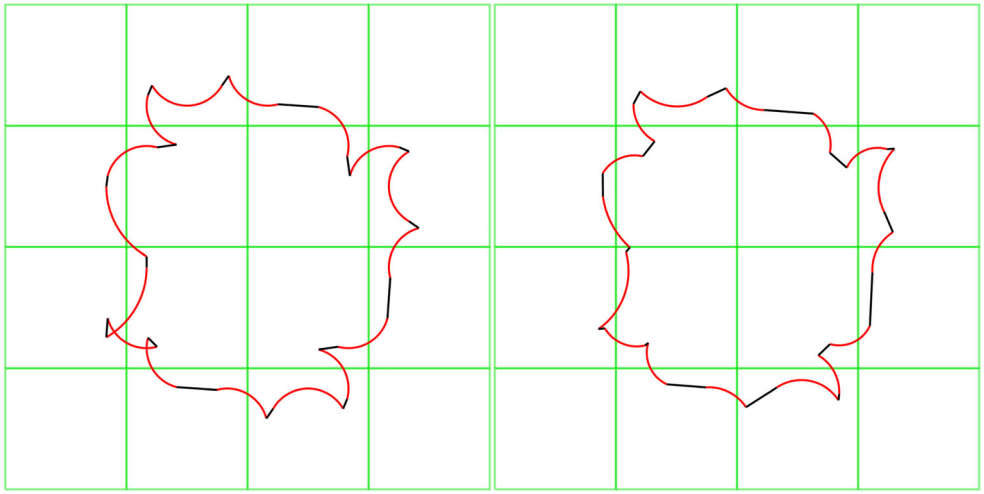


Fig. 14 Solved problem `tiles_suf/4x4_1920`

(a) before DenseOpt (b) after DenseOpt

Their values in Table 3 used for V_{val} generation are taken from [23], where the experiments were carried out in an area of cca 436 m^2.

In conclusion, the proposed set reduction technique is necessary for solving instances of reasonable size. It has only a negligible effect on the solution quality, compared to planning with unreduced data.

3.5 DenseOpt

This subsection documents the performance of the DenseOpt optimization described in Section 2.7. DenseOpt is performed once the GLNS_{arc} finishes planning on the discretely defined problem and attempts to improve the tour weight by sampling new previously unconsidered vertices in a close neighborhood of the vertices present in the tour.

Figure 14 shows the solved problem `4x4_1920` before and after performing DenseOpt. In this particular case, DenseOpt improves the tour weight by about 18%. Interestingly, it also almost entirely eliminates the mutual crossing of neighboring circular segments, even though the results were obtained with edge type *line*.

DenseOpt is run until there is no improvement in the tour weight and has only one parameter N_s. This parameter determines how many times is every vertex resampled in each DenseOpt iteration. Figure 15 shows the progress of tuning this parameter on the dataset `patterns` in the fast mode, where the relative mean tour weight improvement is plotted against the value of N_s. The improvement is calculated as $\frac{w(T)-w_{dense}(T)}{w(T)}$; here, $w(T)$ is the original mean tour weight and $w_{dense}(T)$ is the mean weight after performing DenseOpt. The plot shows the seemingly logarithmic growth of the relative improvement w.r.t. N_s. However, the logarithmic interpolation plotted along the data reveals that the improvement tends to slow down and lies below the interpolating function for $N_s \geq 250$. This is not surprising, as the relative improvement is bounded to be less than 100%, while the limit of a logarithm on an arbitrary base is infinity. Figure 16 shows the time requirements of the DenseOpt w.r.t. N_s. The dependency turns out to be linear, thus the higher value of N_s (number of resampling attempts per vertex) does not accelerate the convergence of the DenseOpt optimization towards a local optimum. Instead, it presumably enables to reach the local optimum in the continuous domain with higher precision, thus resulting in a slightly better final score.

Based on these observations, the parameter value is set to $N_s = 300$ in the following experiments, which corresponds to a mean improvement of circa 7% and an average duration of 1 second on the `patterns` dataset.

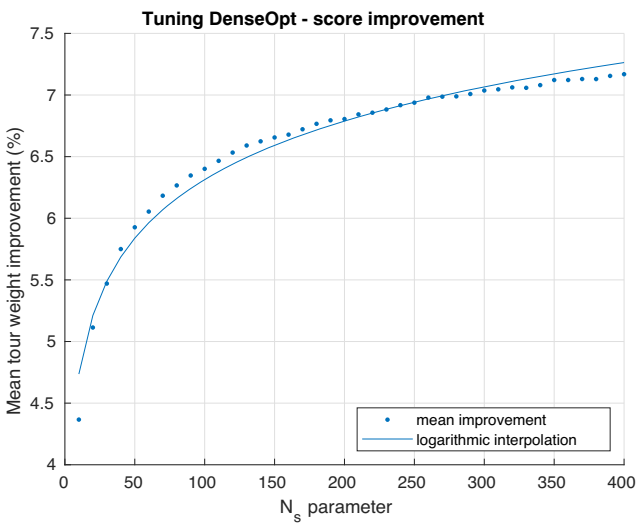


Fig. 15 Tuning DenseOpt - score

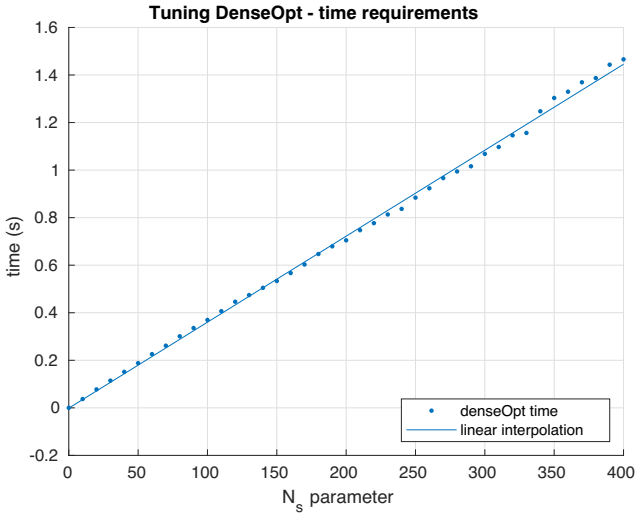


Fig. 16 Tuning DenseOpt - time

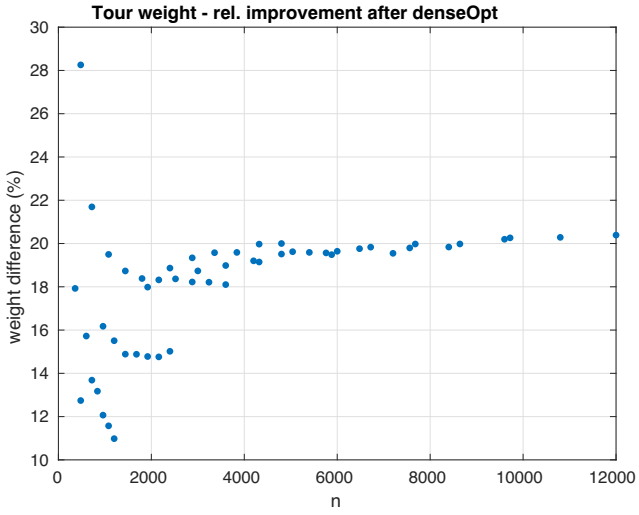


Fig. 18 Tuned DenseOpt performance - relative

DenseOpt performance after parameter tuning is evaluated on the dataset `tiles_suf`. Figure 17 shows the final tour weight before and after DenseOpt across the whole dataset. It can be observed that the plotted statistical properties (minimum, maximum, and standard deviation) are not affected by the DenseOpt in terms of distance from the mean. Figure 18 then shows the relative improvement. The improvement is at least 10%, and it approaches 20% with increasing problem size. When averaged across the whole dataset, the mean improvement slightly exceeds 18%. In contrast, the mean improvement on the `patterns` dataset shown in Fig. 15 in only about 7%. Therefore, the effect of DenseOpt is heavily dependent on the problem structure. Individual sets in the `tiles_suf` problems are placed close together, whereas the sets in the `patterns` dataset are often sparsely distributed across a larger area in small clusters and thus smaller improvements can be achieved through local resampling of vertices.

Finally, Fig. 19 shows the time requirements of DenseOpt compared to the GLNS_{arc} planning time w.r.t. m. DenseOpt times are interpolated with a quadratic function; thus, the time complexity can be estimated as $O(m^2)$, given that all sets are the same size. The same applies to the planning time, which is $O(mn)$, therefore $O(m^2)$ for fixed size sets. For smaller instances, DenseOpt requires more time than planning. At 32 sets, the time requirements of planning and DenseOpt are both circa 2.5 seconds, and for larger instances, planning time becomes dominant.

It was shown in Section 3.3, that solving the `tiles_suf` instances in slow mode yields at most by 2.5% better score than the fast mode and by 1.5% than the default mode. In both cases, the improvement attainable by DenseOpt

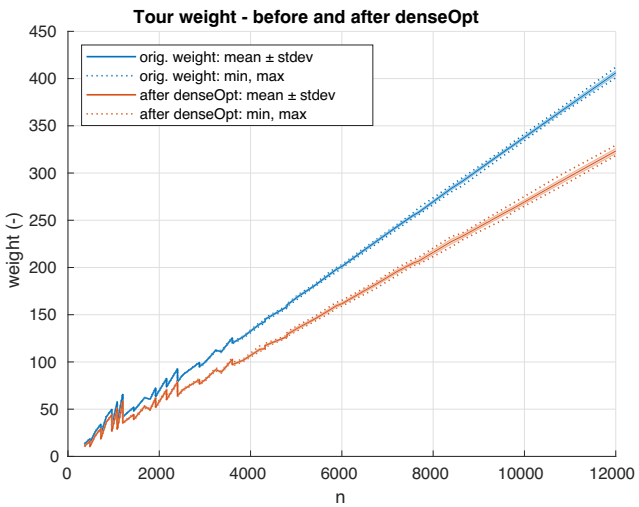


Fig. 17 Tuned DenseOpt performance - absolute

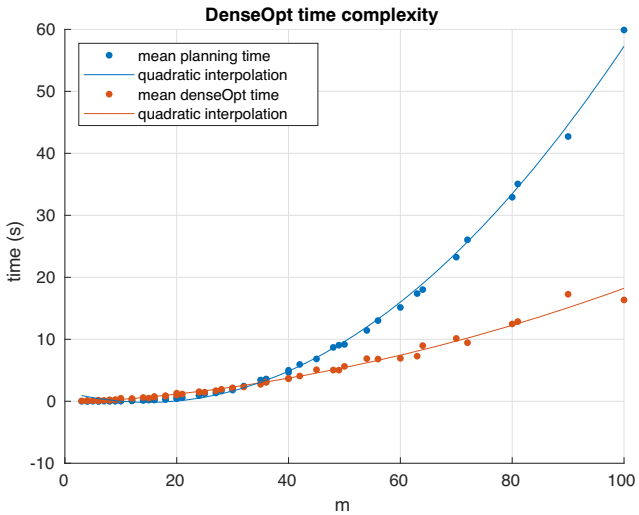


Fig. 19 Tuned DenseOpt time requirements

is several times higher and obtained at a fraction of the additional planning time required by a slower planner mode. Therefore, using the DenseOpt is highly beneficial, as it enables to fundamentally reduce the computation time dedicated for solving the underlying discrete optimization problem and obtain a better quality solution in the continuous domain.

3.6 Set reduction for various edge types

The procedure for sampling valid vertices described in Section 2.6.1 produces a set of 3824 vertices, denoted as V_{val}. This set was then reduced to V_{suf} (Section 2.6.2), a sufficient set of valid vertices generated for edge type $line$. Using V_{suf} instead of V_{val} was then experimentally shown to be highly beneficial in Section 3.4, while planning with the edge type $line$. This subsection presents the results of the set reduction for the edge type $lineWA$ with various angle-weighting constants k.

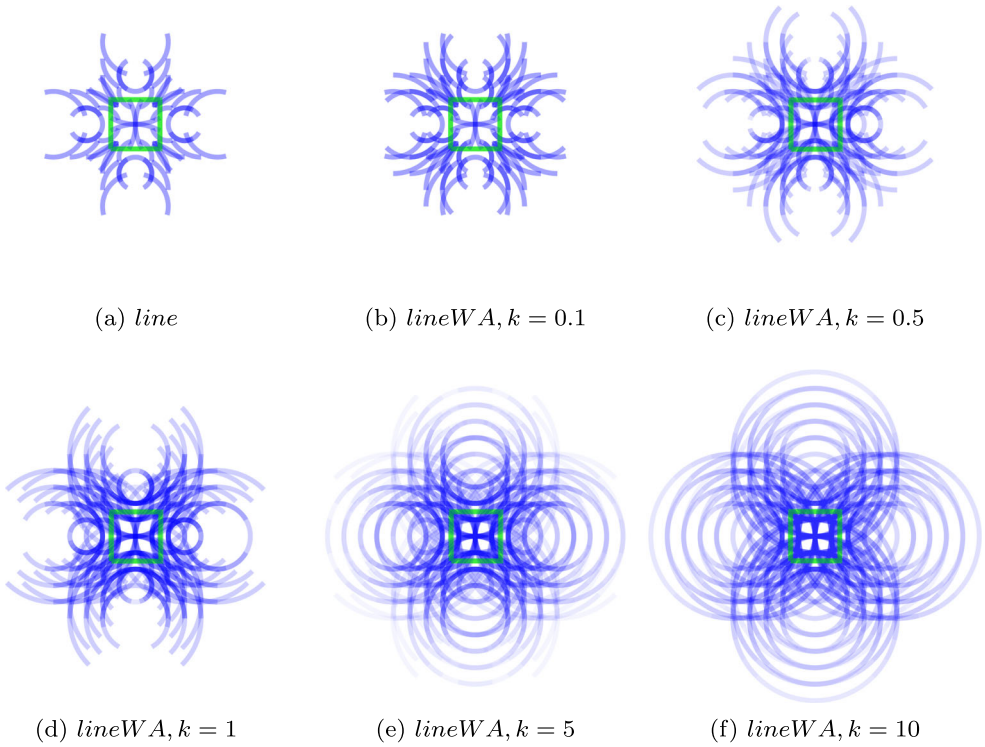
While generating V_{suf} for edge type $line$, the reduction procedure terminated as proposed, i.e., when there was no further increase in the set size after refining both parameters $size$ and $posRes$ of the reduction grid (Section 2.6.2). In the case of $lineWA$, the set reduction procedure turned out to be excessively time-consuming, as the reduction grid has one more parameter $angleRes$ and is effectively 3-dimensional. Therefore, the reduction for $lineWA$ variants was terminated after $posRes$ was refined to 0.5 meter and $angleRes$ remained at initial value $\frac{\pi}{6}$.

Table 5 V_{suf} size for various edge types

Edge type	Angle weight k	V_{suf} size
$line$	0	120
$lineWA$	0.1	184
$lineWA$	0.5	416
$lineWA$	1	634
$lineWA$	5	2366
$lineWA$	10	3684

The generated sets are shown in Fig. 20 and their sizes given in Table 5. Figure 20 shows the previously used V_{suf} generated for edge type $line$. Figure 20b-f show the reduced set for $lineWA$ with various values of angle weight k. It can be observed that the generated set size increases together with k. For $k = 0.1$, the generated set contains 184 vertices, whereas for $k = 10$, it contains 3684 vertices out of 3824 vertices originally present in V_{val}. An explanation for this trend is that the $lineWA$ edge is weighted according to (4). As the angles δ_i and δ_j are assigned greater weight k, the influence of the distance between neighboring vertex endpoints and vertex length decreases. The edge weights then no longer correspond to Euclidean distances, and every vertex is potentially usable given that its endpoints are suitably oriented. Thus, no significant reduction can be achieved for high values of k.

In conclusion, the proposed set reduction technique enables a significant reduction of the vertex sets, which is



(a) $line$ (b) $lineWA, k = 0.1$ (c) $lineWA, k = 0.5$

(d) $lineWA, k = 1$ (e) $lineWA, k = 5$ (f) $lineWA, k = 10$

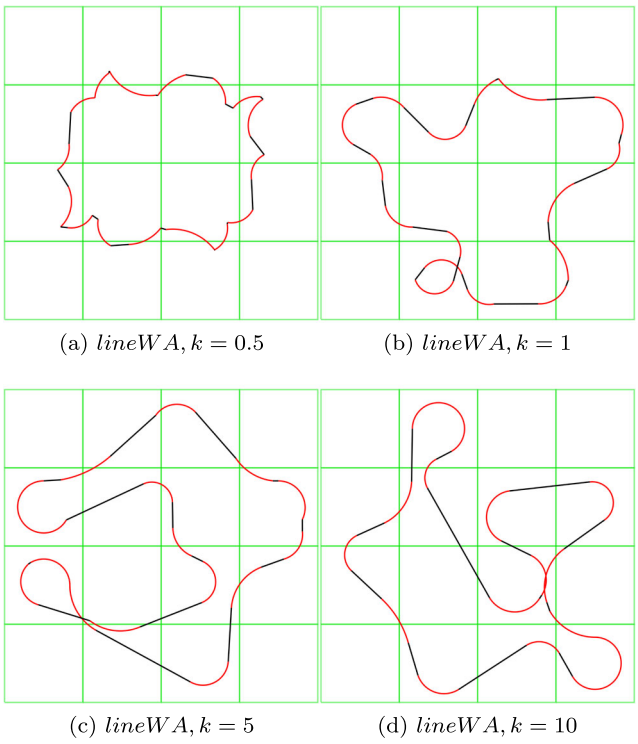
Fig. 20 V_{suf} for various edge types

crucial for solving larger instances. However, the reduction is most effective for metric edge weighting.

3.7 Planning with various edge types

Planning with the edge type $lineWA$ was tested on one problem from the dataset `tiles_suf` and on the whole `patterns` dataset. These datasets are based on the V_{suf} set generated for the edge type $line$. In a real application, problems should be based on V_{suf} generated for the edge type used later in planning. However, creating a separate dataset for each edge type would distort the following comparison of planner performance, as the equivalent problems in different datasets would greatly vary in the number of vertices.

Figure 21 shows the problem `tiles_suf/4x4_1920` solved with four different angle weights k in edge type $lineWA$ after performing DenseOpt. In the case of the smallest value $k = 0.5$, the final tour does not differ much from the solution for edge type $line$, which is shown in Fig. 14. With the increasing value of the constant k, the tour is being straightened at the cost of increasing its Cartesian length, as the algorithm minimizes costly turning maneuvers. For $k = 10$, the black straight line segments (edges) smoothly connect to the red circular segments (vertices), and the trajectory resembles a Dubins path, even though this property is generally not guaranteed.



(a) $lineWA, k = 0.5$ (b) $lineWA, k = 1$

(c) $lineWA, k = 5$ (d) $lineWA, k = 10$

Fig. 21 Solved `4x4_1920` with various edge types

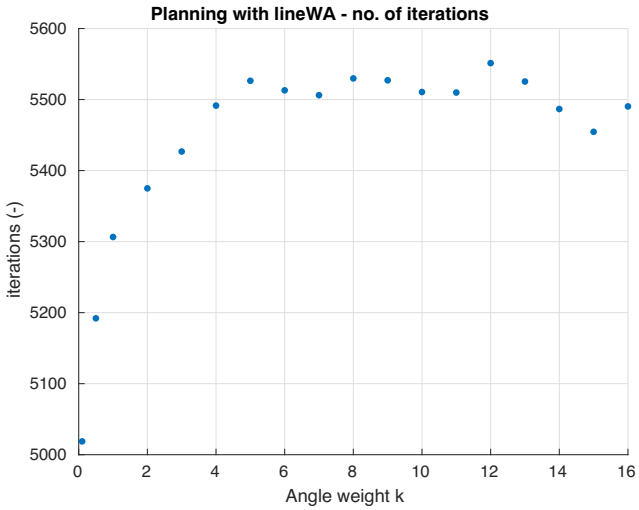


Fig. 22 Planning time for various edge types

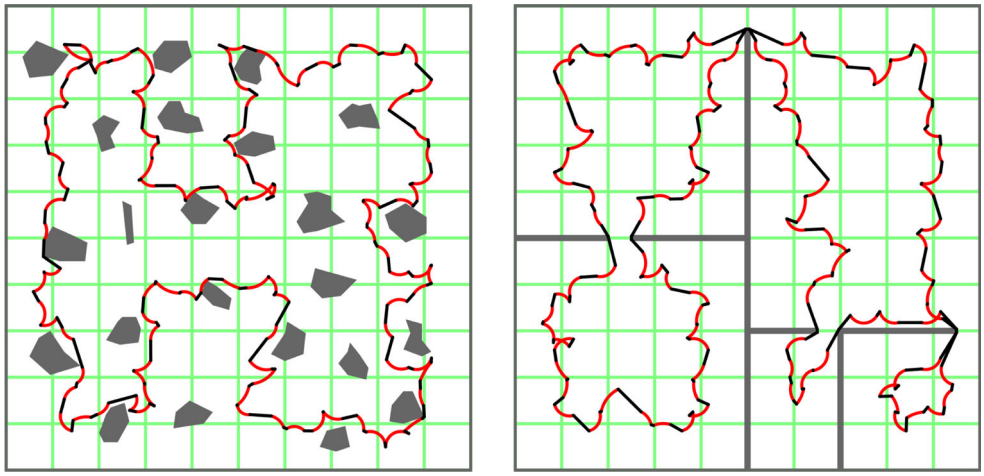
Another interesting trend is revealed in Fig. 22, which shows the mean number of iterations needed per problem averaged over the whole `patterns` dataset for different values of k. All problems in the `patterns` dataset have similar size, ranging between 24 and 32 sets of 120 vertices. The mean number of iterations starts at circa 5000 for $k = 0.1$ and increases slightly above 5500, where it settles for $k > 5$. The GLNS_{arc} terminates after a fixed number of nonimproving iterations in each warm restart, and the results presented indicate that for higher values of k, more iterations are needed to achieve that point. In other words, the local optimum is more difficult to reach.

An explanation for this is that the sets in the dataset `patterns` are spatially clustered, and the edge weights are close to Euclidean distances for small values of k. Both of these properties gradually cease to apply, and for $k > 5$, the edge weights are determined primarily by the mutual orientation of vertex endpoints. Thus, the GTSP_{arc} instances become nonmetric, i.e., they do not satisfy the triangle inequality. Thus, the problems are more difficult to solve, and the planning time increases proportionally to the number of iterations.

In summary, planning with various edge types can be used to produce smooth trajectories, although without guarantee. Using nonmetric edge weighting increases the computational requirements, but not significantly.

3.8 Planning with obstacles

The experimental work [24] motivating the development of GLNS_{arc} assumed that the terrain is obstacle-free, as its primary focus was on determining the accuracy of the STE. This assumption is generally too strong for practical deployment. However, GLNS_{arc} can be directly used for



(a) outdoor map (b) indoor map

Fig. 23 Planning with obstacles

planning in an environment with obstacles, given that a map of the environment is available. There are two extra steps needed before the actual planning, both concerning the preparation of input data.

First, vertices colliding with obstacles in the map must be removed from the instance. Second, the edges connecting the vertices must be planned to avoid the obstacles. For both steps, the VisiLibity1 [29] C++ library was used. This library allows for collision checking and shortest path planning in a provided polygonal map. A single edge then corresponds either to a straight line or to a sequence of multiple straight lines avoiding obstacles, whereas its weight is determined as the length of the line or the sequence of lines. In the following examples, the angle weighting constant k is set to zero; thus, sharp turning is not penalized here.

Figure 23a shows a planned path on an outdoor map `potholes` from the dataset [20]. Similarly to the previous figures, the black segments correspond to edges, red segments to vertices (circular arcs, where the measurements are taken), and green squares to subregions covered by a single vertex. The grey polygons then correspond to an impassable terrain. It is assumed here that these polygons do not affect the radiation propagation. Therefore, the sources can be located anywhere on the map, including the polygons.

Figure 23b shows a planned path in an indoor environment. Here, the walls are considered impassable for radiation. Thus, only such vertices that do not collide with a wall and are located in the same room as the covered green subregion are used for planning.

These examples are meant to illustrate that the GLNS_{arc} is not limited to planning in an obstacle-free outdoor environment. Application in more realistic environments is straightforward and the only additional step is the extraction of invalid vertices from the problem instance.

4 Conclusions

A new planning problem with a background in the search of sources of gamma radiation by a UGV in an outdoor environment was formulated as a GTSP variant and named GTSP_{arc}. GTSP_{arc} is a combinatorial optimization task in the space of maneuvers guaranteeing source detection in preselected regions. To solve this problem in the discrete domain, a state of the art GTSP solver called GLNS was modified and adapted for the application - the new solver is referred to as GLNS_{arc}.

The paper describes all necessary modifications of the GLNS and evaluates the GLNS_{arc} performance in multiple experiments on three generic datasets. These datasets are made publicly available at [38]. Method performance is documented to be sufficient for deployment in the motivating application, both in terms of time requirements and scalability. To achieve this, two additional components are proposed. The first one is a preprocessing technique, which significantly reduces the GLNS_{arc} input data size based on vertex utilization analysis. The technique is shown to have a negligible effect on the solution quality and enables solving instances an order of magnitude larger, thus exploring a larger area. The second one is a postprocessing technique called DenseOpt, which refines the GLNS_{arc} obtained solution in the continuous domain. The DenseOpt proves to be a more time-efficient way of further improving the solution quality than using a slower GLNS_{arc} mode or denser vertex sampling. Moreover, two variants of edge weighting are considered and compared - Euclidean (*line*) and Euclidean with weighted angles (*lineWA*). It is also demonstrated that GLNS_{arc} can be used for planning in the polygonal domain with obstacles. These setups document the applicability of GLNS_{arc} while considering different vehicle models or environments.

The development of $GLNS_{arc}$ was motivated by the experimental work described in [24]. The authors of [24] designed a multirobotic system consisting of a UAV and a UGV, deployed it in real-world experiments, and evaluated the accuracy of the detection. The UAV was used for fast identification of regions of interest, and the UGV subsequently performed accurate localization in the preselected regions. However, planning the UGV trajectory in this scenario relied on a human operator, which does neither guarantee the source detection nor does it produce optimized trajectories. The $GLNS_{arc}$ is designed to automate the planning of the UGV trajectory, while guaranteeing the detection in all preselected regions and producing a near-optimal trajectory. The $GLNS_{arc}$ is unique in this combination of criteria.

Concerning future work - the $GLNS_{arc}$ can be directly applied to planning with splines, Dubins curves, or in environments with obstacles that are not polygonal, given that the trajectory segment weights are precomputed. The main issue arising here is the time demand of the weight precomputing. Comparing these variants thoroughly would provide a valuable insight into the applicability of discrete optimization techniques in similar robot routing problems.

Acknowledgements This work has been supported by the European Regional Development Fund under the project Robotics for Industry 4.0 (registration no. CZ.02.1.01/0.0/0.0/15 003/0000470). The work of David Woller has been also supported by the Grant Agency of the Czech Technical University in Prague, grant SGS18/206/OHK3/3T /37.

Declarations

Competing interests The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Arain MA et al (2015) Global coverage measurement planning strategies for mobile robots equipped with a remote gas sensor. In: Sensors (switzerland), vol 15.3, pp 6845–6871. issn: 14248220, https://doi.org/10.3390/s150306845
2. Bourne JR, Pardyjak ER, Leang KK (2019) Coordinated Bayesian-Based bioinspired plume source term estimation and source seeking for mobile robots. IEEE Trans Robot 35.4:967–986. issn: 19410468. https://doi.org/10.1109/TRO.2019.2912520
3. Chen W, Liu L (2019) Pareto monte carlo tree search for multi-objective informative planning. In: Robotics: Science and systems XV. https://doi.org/10.15607/rss.2019.xv.072
4. Christie G et al (2017) Radiation search operations using scene understanding with autonomous UAV and UGV. J Field Robo 34.8:1450–1468. issn: 15564959. https://doi.org/10.1002/rob.21723
5. De Geer LE (2004) Currie detection limits in gamma-ray spectroscopy. Appl Radiat Isotopes 61.2-3:151–160. issn: 09698043. https://doi.org/10.1016/j.apradiso.2004.03.037
6. Drexl M, Gutenberg J (2012) On the generalized directed rural postman problem. Tech. rep Gutenberg School of Management and Economics
7. Ebenezer J, Murty S (2016) Deployment of wireless sensor network for radiation monitoring. In: 2015 International conference on com- puting and network communications (coconet 2015). Institute of Electrical and Electronics Engineers Inc., pp 27–32. isbn: 9781467373098. https://doi.org/10.1109/CoCoNet.2015.7411163
8. Ferri G et al (2007) Explorative particle swarm optimization method for gas/odor source localization in an indoor environment with no strong air-flow. In: 2007 IEEE International conference on robotics and biomimetics, ROBIO. IEEE computer society, pp 841–846. isbn: 9781424417582. https://doi.org/10.1109/ROBIO.2007.4522272
9. Fischetti M, González JJS, Toth P (1997) A branchand-cut algorithm for the symmetric generalized traveling salesman problem. Oper Res 45.3:378–394. issn: 0030364x. https://doi.org/10.1287/opre.45.3.378
10. Fischetti M, González JJS, Toth P (1995) The symmetric generalized traveling salesman polytope. Networks 26.2:113–123. issn: 10970037. https://doi.org/10.1002/net.3230260206
11. Gabrlik P, Lazna T (2018) Simulation of gamma radiation mapping using an unmanned aerial system. In: IFAC-Papersonline 51.6:256–262. issn: 24058963. https://doi.org/10.1016/j.ifacol.2018.07.163
12. Gutin G, Karapetyan D (2010) A memetic algorithm for the generalized traveling salesman problem. Natural Comput 9.1:47–60. issn: 15677818. https://doi.org/10.1007/s11047-009-9111-6. arXiv:0804.0722
13. Han J et al (2013) Low-cost multi-UAV technologies for contour mapping of nuclear radiation field. J Intell Robot Syst: Theory Appl 70.1-4:401–410. issn: 09210296. https://doi.org/10.1007/s10846-012-9722-5
14. Helsgaun K (2000) An effective implementation of the Lin-Kernighan traveling salesman heuristic. Eur J Oper Res 126:106–130
15. Helsgaun K (2013) GTSP problem libraries BAF, MOM and GTSP+. http://akira.ruc.dk/~keld/research/GLKH/. accessed 2020-02-03
16. Helsgaun K (2015) Solving the equality generalized traveling salesman problem using the Lin-Kernighan-Helsgaun Algorithm. Math Programm Comput 7.3:269–287. issn: 18672957. https://doi.org/10.1007/s12532-015-0080-8
17. Hollinger G, Sukhatme G (2016) Sampling-based motion planning for robotic information gathering. In: Robotics: Science and systems. https://doi.org/10.15607/rss.2013.ix.051
18. Hoos HH, Thomas S (2014) On the empirical scaling of run-time for finding optimal solutions to the travelling salesman problem. Eur J Oper Res 238.1:87–94. issn: 03772217. https://doi.org/10.1016/j.ejor.2014.03.042
19. Isaacs JT, Hespanha JP (2013) Dubins traveling salesman problem with neighborhoods: A graph-based approach. Algorithms 6.1:84–99. issn: 19994893. https://doi.org/10.3390/a6010084. http://www.mdpi.com/1999-4893/6/1/84
20. Kalisiak M, Faigl J (2013) Motion planning maps - dataset. http://agents.fel.cvut.cz/~faigl/planning/. accessed 2020-07-07
21. Laporte G, Asef-Vaziri A, Sriskandarajah C (1996) Some applications of the generalized travelling salesman problem. J Oper Res Soci 47.12:1461–1467. issn: 14769360. https://doi.org/10.1057/jors.1996.190

22. Laporte G, Nobert Y (1983) Generalized traveling salesman problem through n sets of nodes: an integer programming approach. INFOR: Inf Syst Oper Res 21.1:61–75. issn: 03155986. https://doi.org/10.1080/03155986.1983.11731885
23. Lazna T Optimizing the localization of gamma radiation point sources using a UGV. In: 2018 ELEKTRO Conference Proceedings. Institute of Electrical and Electronics Engineers Inc., pp 1–6. (2018) https://doi.org/10.1109/ELEKTRO.2018.8398368
24. Lazna T, et al. (2018) Cooperation between an unmanned aerial vehicle and an unmanned ground vehicle in highly accurate localization of gamma radiation hotspots. Int J Adv Robot Syst 15.1:172988141775078. issn: 17298814. https://doi.org/10.1177/1729881417750787
25. Lilienthal A, Loutfi A, Duckett T (2006) Airborne chemical sensing with mobile robots. Sensors 6.11:1616–1678. issn: 1424-8220. https://doi.org/10.3390/s6111616
26. Liu Z, Abbaszadeh S, Sullivan CJ (2018) Spatial-temporal modeling of background radiation using mobile sensor networks. PLOS one, vol 13. Ed. by Raghuraman Mudumbai. issn, pp 1932–6203. https://doi.org/10.1371/journal.pone.0205092
27. Miller A., Machrafi R., Mohany A. (2015) Development of a semi-autonomous directional and spectroscopic radiation detection mobile platform. Radiat Measur 72:53–59. issn: 13504487. https://doi.org/10.1016/j.radmeas.2014.11.009
28. Noon CE, Bean JC (1993) An efficient transformation of the generalized traveling salesman problem. INFOR: Inf Syst Oper Res 31.1:39–44. issn: 0315-5986. https://doi.org/10.1080/03155986.1993.11732212
29. Obermeyer KJ, Contributors (2008) VisiLibity: A C++ Library for Visibility Computations in Planar Polygonal Environments. http://www.VisiLibity.org. accessed 2020-07-07
30. Pop PC. (2007) New integer programming formulations of the generalized travelling salesman problem. Amer J Appl Sci 4.11:932–937. issn: 15543641. https://doi.org/10.3844/ajassp.2007.932.937
31. de Julio Rozental J (2002) Two decades of radiological accidents direct causes, roots causes and consequences. Braz Arch Biol Technol 45.spe:125–133. issn: 1516-8913. https://doi.org/10.1590/s1516-89132002000500018
32. Smith SL, Frank I (2017) GLNS An effective large neighborhood search heuristic for the generalized traveling salesman problem. Comput Oper Res 87:1–19
33. Soin PK et al (2019) Application of a novel search method to handheld gamma radiation detectors. IEEE Sens J:1–1. issn: 1530-437X. https://doi.org/10.1109/jsen.2019.2945314
34. Uher J. et al (2007) Directional radiation detector. IEEE Nuclear Sci Symp Conf Rec 2:1162–1166. isbn: 1424409233. https://doi.org/10.1109/NSSMIC.2007.4437213
35. Wendorf M (2020) Broken Arrows - The World's Lost Nuclear Weapons. shorturl.at/ryBLO
36. Wheatley S, Sovacool BK, Sornette Didier (2016) Reassessing the safety of nuclear power. Energy Res Social Sci 15:96–100. issn: 22146296. https://doi.org/10.1016/j.erss.2015.12.026
37. Wiedemann T, Shutin D, Lilienthal AJ (2019) Modelbased gas source localization strategy for a cooperative multi-robot system— A probabilistic approach and experimental validation incorporating physical knowledge and model uncertainties. Robot Auton Syst 118:66–79. issn: 09218890. https://doi.org/10.1016/j.robot.2019.03.014
38. Woller D (2019) GTSP with arcs - 3 datasets. http://imr.ciirc.cvut.cz/Datasets/GTSP-arc. accessed 2020-02-03
39. Zakaria AH et al (2017) Development of autonomous radiation mapping robot. In: Procedia Computer Science. vol 105. Elsevier B.V., pp 81–86. https://doi.org/10.1016/j.procs.2017.01.203

David Woller is a Ph.D. student at the Czech Institute of Informatics, Cybernetics, and Robotics, Czech Technical University (CTU) in Prague. He obtained his Master's degree in 2019 in Cybernetics and Robotics at CTU in Prague. He spent 3 months at a research internship at the University of Avignon, Laboratory of Informatics, France. His main research interests are route planning in robotic applications, metaheuristics, and local search techniques in combinatorial optimization.

Miroslav Kulich is an assistant professor at the Czech Institute of Informatics, Cybernetics, and Robotics, Czech Technical University (CTU) in Prague. He received a Ph.D. degree in Artificial Intelligence and Biocybernetics at CTU in Prague, and an RNDr. degree at Charles University in Prague, Faculty of Mathematics and Physics. He also spent 6 months at a research fellowship at Helsinki University of Technology, Automation Technology Laboratory, Finland. Miroslav Kulich serves as a reviewer for several impacted journals as well as a program committee member of relevant international conferences. His research interests include planning for single and multi-robot systems, especially in exploration and search&rescue scenarios.