10.24425/acs.2020.135845

Archives of Control Sciences
Volume 30(LXVI), 2020
No. 4, *pages 653–666*

Algorithm for solving the Discrete-Continuous Inspection Problem

R. GRYMIN, W. BOŻEJKO, Z. CHACZKO, J. PEMPERA and M. WODECKI

The article introduces an innovative approch for the inspection challenge that represents a generalization of the classical Traveling Salesman Problem. Its priciple idea is to visit continuous areas (circles) in a way, that minimizes travelled distance. In practice, the problem can be defined as an issue of scheduling unmanned aerial vehicle which has discrete-continuous nature. In order to solve this problem the use of local search algorithms is proposed.

Key words: discrete-continuous optimization, UAV, VTOL, autonomous drone, zero emission

1. Introduction

In recent years, unmanned aerial vehicles (UAV) have become very popular in many areas. Inspection is one of the applications. Their advantage over terrestrial vehicles is independency of a finite number of routes, due to the fact they do not participate in traffic jams and can perform inspections without having to continously interact with humans. Manual control of an unmanned aerial vehicle is associated with the danger of possible operator errors – in order to eliminate these errors, autonomous inspection algorithms are used.

R. Grymin, W. Bożejko (corresponding author) and J. Pempera are with Department of Control Systems and Mechatronics, Wrocław University of Science and Technology, Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland, e-mails: {radoslaw.grymin, wojciech.bozejko, jaroslaw.pempera}@pwr.edu.pl.

Z. Chaczko is with Faculty of Engineering and Information Technology (FEIT), University of Technology, Sydney(UTS), Australia, NSW, Sydney, e-mail: zenon.chaczko@uts.edu.au.

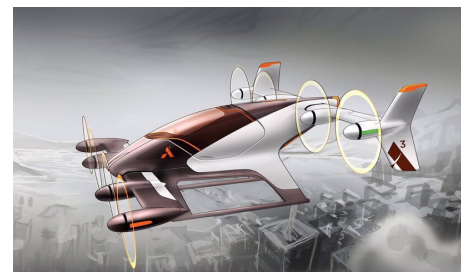
M. Wodecki is with Department of Telecommunications and Teleinformatics, Wrocław University of Science and Technology, Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland, e-mail: mieczyslaw.wodecki@pwr.edu.pl.

The paper was partially supported by the National Science Centre of Poland, grant OPUS no. 2017/25/B/ST7/02181 and statutory grant no. 0401/0023 /18 of Faculty of Electronics, Wrocław University of Science and Technology.

Received 10.08.2019. Revised 10.05.2020.

A serious challenge connected with using an electric drive is a relatively short flight time compared to the time required to fully charge its UAV battery. For example, one of the most modern UAVs DJI Matrice 200 (Fig. 1a), widely used in inspection, can fly from 13 up to 38 minutes until its complete discharge (depending on battery used, mounted equipment and speed). An alternative to UAV in the future will be the use of small VTOL (Vertical Take-Off and Landing) planes that allow its users not only for vertical takeoff but also for vertical landing. An example of an autonomous VTOL is the Airbus A3 Vahana [16] shown in Fig. 1b.

(a) DJI Matrice 200
Source: dji.com



(b) Airbus A3 Vahana.
Source: vahana.aero

Figure 1: Examples of aerial vehicles enabling inspections

The use of an autonomous flight system in a vehicle, in the future, makes it available to people who do not have a professional helicopter pilot license (as on 31/12/2017 only 278 people had such a license in Poland [5]). The vehicle has an electric drive, thanks to which it has zero emissions and is so quiet that it can carry out inspections in the city without exceeding the noise standards. While writing this article, Airbus A3 Vahana was still in the testing phase.

The use of UAV or VTOL is associated with the problem of long battery charging. Therefore, the autonomous inspection requires the use of effective flight planning algorithms. In this article an effective algorithm for solving the problem of autonomous inspection is presented. In Section 2, the concept of Inspection Problem is introduced as a challenge that can be solved by applying a novel two-level optimization solution strategy. At the first level, the fast Powell's method is used to optimize the flight path, and at the second level, a higher level optimization procedure based on the 2-Opt and the Simulated Annealing algorithms is introduced. Section 3 discusses details of experimental research conducted, whereas Section 3.1 covers a description of the algorithm that was used for generation of the presented Inspection Problem instance. Section 3.2 contains a description of computational experiments that were conducted. Finally, Section 4 summarizes the results of the conducted research work.

2. Inspection problem

In the problem of data inspection there are o objects from the set $O = \{1, 2, 3, \ldots, o\}$, for which the inspection must be conducted. For each object $i \in O$ there is defined three-dimensional subspace, from which clear pictures can be taken. This space will be brought closer with the use of a sphere. The optimization problem is to find the shortest path of the aircraft, which passes through all areas of visibility at least once (in order to take pictures). The flying object starts from a certain starting point and must return to it after the inspection.

This problem is a generalization (continuous version) of *Set TSP*, also known as: *Generalized TSP* (*GTSP*), *International TSP*, *Group TSP, One-of-a-Set TSP, Coverng Salesman problem* or *Multiple Choice TSP*. The problem is strongly NP-hard as it can be reduced to *TSP* when the areas are single points. *GTSP* appears frequently in traffic planning problems (Imeson and Smith [9]; Mathew et al. [11]; Wolff et al. [18]). For this class of problems, many heuristics and exact algorithms were proposed in the literature, including evolutionary algorithms (Snyder and Daskin [15]), also in the parallel version (Bożejko and Wodecki [4]). Thus far, there is a limited, if any, research material or literature that covers the continuous version of *GTSP*.

In further considerations it is assumed that the visibility space of the object being inspected can be approximated by means of a hemisphere (for example in the case of photographing objects and transmitter – radio receiver communication) and that the unmanned aircraft flies at a fixed ceiling and has no kinematic limitations, i.e. it can navigate any curves as it is a holonomic robot. Then the inspection area for each inspection object can be described with the use of a sphere. This geometrical object will be called *the visibility circle*, whilst the point at which the inspection (e.g., photographing) is performed will be called the *the inspection point*.

For each $i \in O$ object, there is a visibility sphere (space) specified with triangle (x_i, y_i, r_i) given, where x_i and y_i denote the center coordinates, while r_i ($r_i > 0$) denotes the radius of the visibility sphere of the object. Before the flight begins, the unmanned vehicle is at the point with coordinates (x_0, y_0). After the inspection, the unmanned vehicle must return to this point. The purpose of the optimization is not only to determine the inspection point for each object but also to find the order of visiting inspection points for which the length of the distance traveled by the unmanned aircraft is the shortest. To illustrate the problem, a sample solution acceptable for a certain problem instance is presented in Figures 2 and 3.

The object's inspection point $i \in O$ will be denoted by a pair (X_i, Y_i), which of course must belong to the visibility circle (x_i, y_i, r_i). The order of visiting inspection areas can be described using sequences $S = (0, S_1, S_2, \ldots, S_o, 0)$,

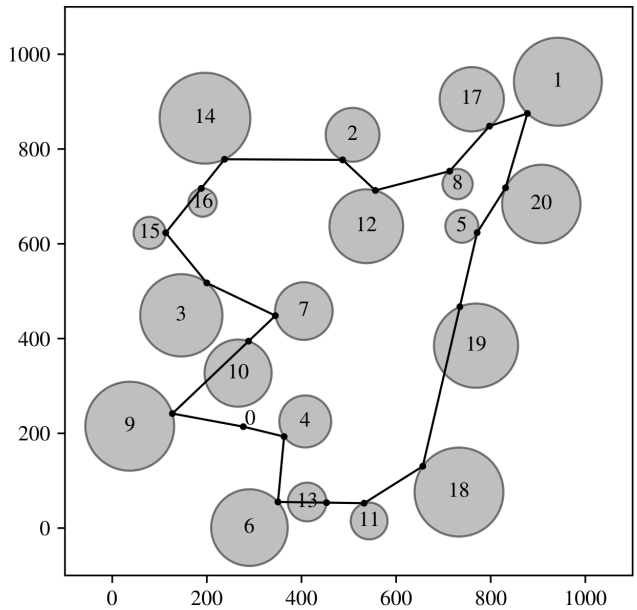


Figure 2: Illustration of disjoint visibility circles, starting point 0 (zero) and path found by the algorithm

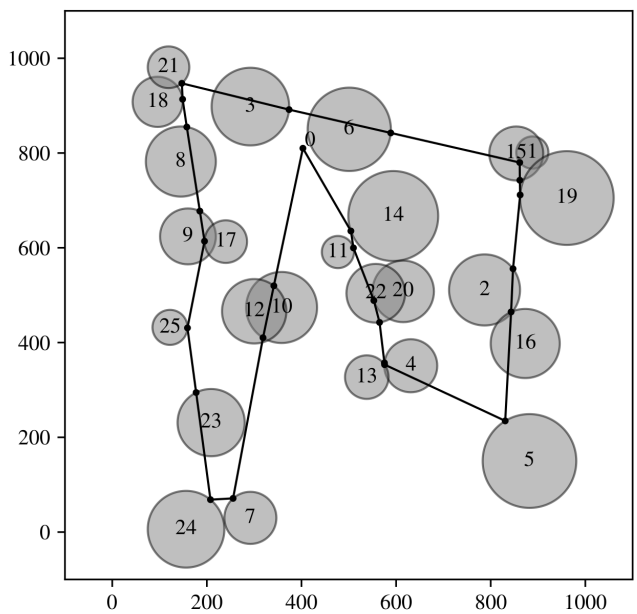


Figure 3: Illustration of the overlapping circles of visibility, starting point 0 (zero) and the inspection plan found by the algorithm

where $S_i \in O$ denotes the inspection object, while 0 (zero) denotes the starting point.

For the given inspection points of the (X_i, Y_i), $i \in O$ objects, and the order of visiting the areas $S = (0, S_1, \ldots, S_o, 0)$ the length of the flight path can be determined from the formula

$$L(S, X, Y) = \sum_{s=1}^{o+1} d\left(X_{S_{s-1}}, Y_{S_{s-1}}, X_{S_s}, Y_{S_s}\right), \tag{1}$$

where $d(X_k, Y_k, X_l, Y_l)$ is the Euclidean distance between points (X_k, Y_k) and (X_l, Y_l). In the optimization problem being considered there are two types of decision variables:

1) S – sequence of visiting inspection areas (permutation, see e.g. [3]),

2) X, Y – inspection points in each area (real variables).

Determining the shortest flight path of a flying object requires designing algorithms combining the characteristics of continuous and discrete optimization algorithms.

Before introducing the proposed optimization algorithm, we will present a certain property of the problem allowing to reduce the size of the problem (the number of considered inspection areas) without losing the possibility of finding an optimal solution for the primary problem.

Lemma 1 *Let* (x_i, y_i, r_i) *be a circle contained in the circle* (x_j, y_j, r_j) *i.e. there is*

$$d(x_i, y_i, x_j, y_j) + r_i \leqslant r_j$$

then the circle (x_j, y_j, r_j) *can be omitted during the path determination.*

Proof. Let us assume that $r_i \leqslant r_j$, and that a circle with radius r_i is contained in a circle with radius r_j. Then, by visiting a certain point (X_i, Y_i) belonging to the smaller circle (x_i, y_i, r_i), we visit simultaneously the point which belongs to the biggest circle (x_j, y_j, r_j), which ends the proof. □

2.1. Optimization algorithm

In the optimization problem under consideration, there should be two components determined: the order in which inspection areas are visited and in each area – the inspection point. Both the selection of the order and position of the inspection points affects the length of the flight path of the flying object (see Eq. (1)). To determine the shortest path of a flying object, we propose a hierarchical algorithm in which the high level is responsible for determining the sequence of visited areas, while the low level is responsible for finding the inspection points

for a given sequence, so that the path length is the shortest possible. The aim of the high level is to find a sequence for which the route length determined by the low level algorithm is the shortest.

Before discussing the algorithm for designating inspection points for a given sequence of S of visiting areas, we must notice that

Lemma 2 *The feasible flight path of a flying object must have at least one point in common with the circumference of each visibility circle.*

For proof of lemma 2, it is enough to note that the inspection point must be in the circle of visibility, so if it is not on the circumference, then the flying object first had to go through a point on the circumference and then leave the same or another point on the circumference circuit.

On the other hand, each point on the perimeter of the circle can be clearly described by the angle between the OX axis and a line connecting inspection point with the location of the inspection object. Finally, the object's inspection point $i \in O$ is uniquely specified using the angle α_i. Cartesian coordinates of a point can be determined by means of equations:

$$X_i(\alpha_i) = x_i + r_i \cos(\alpha_i), \tag{2}$$

$$Y_i(\alpha_i) = y_i + r_i \sin(\alpha_i). \tag{3}$$

For a given sequence S, $L(S, X(\alpha), Y(\alpha))$, $\alpha = (0, \alpha_1, ...\alpha_o, 0)$, $X(0) = X_0$, $Y(0) = Y_0$ is non-linear function with o variables. The minimum of the function $L(S, X(\alpha), Y(\alpha))$ can be determined with the use of such algorithms as: Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algortihm (Byrd, Lu and Nocedal [6], Zhu et al. [19]), The Nelder-Mead method (Nelder and Mead [12]), Sequential Quadratic Programming (SQP) method (Kraft [10], Nocedal [13]). Unfortunately, these methods are time-consuming due to the large number of variables (inspection objects). Therefore, to find the shortest path, the greedy algorithm is proposed, in which in each iteration the greatest improvement is obtained by changing the value of one variable (the angle describing the point of visibility).

More precisely, let us consider three consecutive inspection points $(X_a(\alpha_a), Y_a(\alpha_a))$, $(X_b(\alpha_b), Y_b(\alpha_b))$, $(X_c(\alpha_c), Y_c(\alpha_c))$. The angle α_b is locally optimal angle for object b in reference to points $(X_a(\alpha_a), Y_a(\alpha_a))$ and $(X_c(\alpha_c), Y_c(\alpha_c))$ if

$$d(X_a(\alpha_a), Y_a(\alpha_a), X_b(\alpha_b), Y_b(\alpha_b)) + d(X_b(\alpha_b), Y_b(\alpha_b), X_c(\alpha_c), Y_c(\alpha_c)) \tag{4}$$

takes the minimum value. The angle α_b^*, for which the expression (4) takes the minimum value was designated with the Powell's method (Powell [14]).

We determine by

$$\Delta_b = L(S, X(\alpha), Y(\alpha)) - L(S, X(\alpha^*), Y(\alpha^*)), \tag{5}$$

where $\alpha^* = (0, \alpha_1, ..., \alpha_b^*, ..., \alpha_o, 0)$, improving (shortening) the length of the shortest path by optimizing the angle α_b.

In each iteration of the algorithm, an optimal angle is found locally for each inspection object. Then an object is designated for which the angle change generates the greatest improvement. This angle becomes the base angle for this object in the next iteration of the algorithm.

The algorithm stops when the improvement is not greater than the arbitrarily selected value of ϵ or after the execution of $MaxIter$ iterations. The scheme of the proposed method is presented in the Algorithm 1. In order to find the best sequence of visiting inspection areas, one can apply one of many algorithms dedicated to scheduling problems available. However, for the start, it should be noted that the investigated problem is very similar to a well-known Traveling Salesman Problem which aims to find a solution for the optimal order of visiting inspection areas. In the investigated problem, the positions of the inspection points and the resulting distances between the points depend on the sequence of visiting inspection areas whilst in the classical Traveling Salesman Problem (*TSP*) the position of inspection points are fixed.

Algorithm 1: Designation of the shortest path for a given sequence S

Input : α^0 – initial angles vector;
ϵ – precision parameter;
Output : $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_o)$;

```
1 α ← α^0
2 for iter ← 1 to MaxIter do
3     for b ← 1 to o do
4         determine Δ_b
5     determine such k that k = arg max_{s∈O} Δ_s
6     if Δ_k < ε then
7         return α
8     α_k ← α*_k
```

Considering the similarity of the investigated scenario to the Traveling Salesman Problem, not only a 2-Opt algorithm was developed but also an algorithm based on the simulated annealing method was designed, which is one of the most effective methods of constructing algorithms for sequential problems.

2-Opt Algorithm

2-Opt Algorithm (Croes [7]) is one of the simplest and highly effective algorithms for the traveling salesman problem. It is an algorithm that determines the local optimum for the 2-Opt neighborhood. The 2-Opt neighborhood consists of sequences being a modification of the base sequence $S = (0, S_1, ..., S_o, 0)$. Each modification (also called a move) is described by a pair $v = (a, b)$. As a result of the modification of the base sequence described by $v = (a, b)$ is a new route which takes the form

$$S^v = (0, S_1, \ldots, S_{a-1}, S_b, S_{b-1}, \ldots, S_{a+1}, S_a, S_b, \ldots, S_o, 0). \tag{6}$$

In each iteration of the neighborhood algorithm, the best sequence is selected which becomes the base sequence in the next iteration. The algorithm terminates when no better solution is located in the vicinity of the base solution, i.e. the base solution is locally optimal.

Simulated Annealing Algorithm

The simulated annealing algorithm (SA, see e.g. [2, 17]) refers to the thermodynamic process of metal annealing, where the metal sample is subjected to a cooling process to achieve the desired properties (hardness, elasticity, etc.). The idea of operating of SA boils down to generating a trajectory of search (where each successive solution is chosen randomly from the neighborhood of the current solution) based on random moves in the neighborhood. The drawn solution is accepted (it replaces the current solution in the next iteration) unconditionally when it is not worse than the current one. It is also possible to accept worse solutions with a probability depending, for example, on the temperature and the difference in the value of the objective function (the so-called acceptance function). A detailed description of the method can be found in [1]. In the algorithm implemented to solve the problem under consideration, the 2-Opt neighborhood was used. The algorithm performed $MaxIter = 2000$ iterations. As a result, the trajectory of searching for the SA algorithm is carried out in 'statistically good' direction.

Finally, three algorithms were implemented: 2-Opt (C), 2-Opt and SA. In the 2-Opt(C) algorithm, the route length was determined based on the location of the inspection objects, i.e.

$$L(S, x, y) = d(x_0, y_0, x_{S_1}, y_{S_1}) + \sum_{s=1}^{o-1} d(x_{S_s}, y_{S_s}, x_{S_{s+1}}, y_{S_{s+1}}) + \\ + d(x_{S_o}, y_{S_o}, x_0, y_0). \tag{7}$$

3. Experimental research

The results of the research are strongly influenced by the selection of appropriate test instances and the selection of appropriate metrics. The aim of this research was to check the efficiency of the algorithm for a problem that has not been described yet, so there are no test results for other algorithms to compare its efficiency.

3.1. Data generating

The algorithm should be used for any instances of the inspection problem, it is not dedicated to specific instances of the problem (e.g., in which the inspection areas are arranged schematically, according to some rule), hence the test instances were generated in a random manner.

There were 6 instance groups generated with the same number of objects to visit $n \in \{5, 10, 15, 20, 25, 30\}$. Due to the fact that the instances were generated randomly, each instance group contained 10 instances of the problem. Such a procedure was designed to allow us to investigate what the average expected performance of the algorithm for the n instance would be. Each instance is described by the means of centers of circles, the length of their radiuses and coordinates of the base. The instance generating algorithm randomly placed the centers of circles on a square of 1000 side, and the radiuses randomized with the uniform distribution of $U(a, b)$ with the distribution parameters characteristic for each group, $(a, b) \in \{(100, 200), (50, 150), (40, 120), (30, 100), (30, 100), (30, 100)\}$. Higher size instances had to contain smaller circles to get some overlapping circles and some non-overlapping circles. The way of generating data: non-overlapping and overlapping circles is presented in Algorithms 2 and 3, respectively. An algorithm that generates circles that are not overlapping, used to illustrate the case of non-overlapping circles (Figure 2), works on the principle of adding successive circles. It starts its operation from drawing the base coordinates $base_x, base_y$ (each coordinate is drawn with the uniform distribution $U(0, 1000)$). Then, in a loop, he adds more circles in turn. The addition of a new circle consists of two stages: the drawing of the (`center`) and the radius. The center of the circle is drawn by the `computeCenter` function until it is possible to add a circle that has a radius at least equal to `minRadius` and is distant from the other circles and base by at least `minSpaceBetweenCircles`.

When such a measure is found, the function `maximalRadiusForCircle` sets the maximum possible radius of the circle `maximalRadiusForCircle`, so that the distance from the remaining circles and base is not less than `minSpaceBetweenCircles`. At the very end a radius draw is made with the distribution of U (`minRadius`, `maximalRadiusForCircle`).

The algorithm that generates overlapping circles (Algorithm 2) at the beginning draws coordinates of the base (each coordinate is drawn with the distribution

Algorithm 2: The algorithm of generation of overlapping circles

```
1  base_x ← rand() · 1000;
2  base_y ← rand() · 1000;
3  base ← (base_x, base_y);
4  circles ← [ ];
5  for iter = 1 to n do
6  |   x ← rand() · 1000;
7  |   y ← rand() · 1000;
8  |   r ← rand( ) · (b−a) + a;
9  |      {append 3-tuple to the end of circles vector}
10 |   circles ← (circles | [(x, y, r)]);
11 return base, circles;
```

Algorithm 3: The algorithm generating non-overlapping circles

```
1  centers ← [ ];
2  radii ← [ ];
3  base_x ← rand() · 1000;
4  base_y ← rand() · 1000;
5  base ← (base_x, base_y);
6  for iter = 1 to n do
7  |   center ← computeCenter(centers, radii, base, minRadius, areaWidth,
   |     areaHeight, minSpaceBetweenCircles);
8  |   maximalRadiusForCircle ← maximalRadiusForCircle(center, base, centers,
   |     radii, maxRadius, minSpaceBetweenCircles);
9  |   r ← rand( ) · (maximalRadiusForCircle−a) + a;
10 |   radii ← (radii | r);
11 |   centers ← (centers | [center]);
12 return centers, radii, base;
```

of 1000). Next, further circles are generated in the loop. Coordinates of centers are generated randomly with the distribution of $U(0, 1000)$, whereas the radius is generated with the distribution of $U(a, b)$. The tests used instances in which some circles overlapped.

3.2. Computational experiments

2-Opt (C), 2-Opt and SA metaheuristics were programmed in C++ in Visual Studio 10 and run on a single processor core Intel Core i7 3.5 GHz, with 8 GB RAM running under the operating system Windows 7. The SA algorithm was designed in the version with auto-tuning of the initial parameters (see [1]). In order

to compare the effectiveness of individual optimization stages, the (*percentage relative deviation, PRD*) was examined between the length of the path S^A returned by the tested algorithm $A \in \{\text{2-Opt(C), 2-Opt, SA}\}$ and the length of the best found path L^*.

This measurement is defined by the equation:

$$PRD(A) = \frac{L(S^A) - L^*}{L^*} \cdot 100\%. \tag{8}$$

Figure 4 shows the order in which algorithms are run to find an optimized path. The 2-Opt(C) algorithm generates in a very short time the path that is improved by the 2-Opt algorithm and three times by the SA algorithm. After completing each calculation phase, the length of the shortest path is memorized. During the next phase of calculations, the best path from the previous phase becomes the initial path for the next phase.

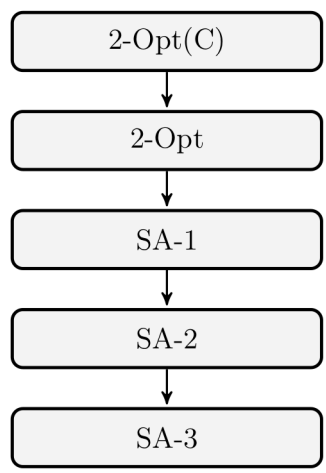


Figure 4: The order of running optimization algorithms

Table 1 presents the percentage relative error PRD of solutions returned by the subsequent phases of the algorithm. This error was determined in reference to the shortest path length L^* obtained in the last run of the SA algorithm (SA-3). It can be seen that the length of the path when the aerial vehicle is approaching the object is much longer than the length of the path returned using the proposed method. It is longer on average by 39.14%, whereas the average value varies between 32.56% and 46.03%. These observations confirm the high effectiveness of the proposed methods of path determination for the aerial vehicle.

It is worth noting that the application of the proposed method to the sequence generated by 2-Opt(C) (see column 2-Opt (C) $L(S)$) significantly reduces the length of the route. The average PRD value is only 4.19% and ranges from

Table 1: Average relative error of the algorithms

n	2-Opt(C) $LC(S)$	2-Opt(C) $L(S)$	2-Opt $L(S)$	SA-1 $L(S)$	SA-2 $L(S)$
5	46.03	1.35	0.00	0.00	0.00
10	37.74	4.09	0.55	0.00	0.00
15	32.56	1.79	0.65	0.43	0.00
20	33.25	4.12	3.15	0.10	0.00
25	43.57	6.71	3.29	1.99	0.03
30	41.69	7.08	3.53	0.96	0.32
Average	39.14	4.19	1.86	0.58	0.06

1.35% to 7.08%. Unfortunately, its efficiency decreases as the number of objects increases.

In the next phases, the algorithms operate on routes in which inspection points are determined by the bottom-level method. In the case of the 2-Opt algorithm, the average error is 1.86% and ranges from 0% to 3.53%. This error increases as the number of objects increases.

4. Summary

The presented study considers a discrete-continuous problem of routing which represents a generalization of the classical *TSP* problem concerning the determination of the unmanned vehicle routing. The test scenarios involved instances where some of the path circles overlap with each other. The proposed cascade, two-level optimization method allowed for a significant improvement in results compared with the cases where the aerial vehicle was flying directly towards the object being photographed, i.e. to the center of the target inspection area.

References

[1] E.H.L. Aarts and P.J.M. van Laarhoven: *Simulated Annealing: A Pedestrian Review of the Theory and Some Applications*, In: Devijver P.A., Kittler J. (eds), Pattern Recognition Theory and Applications, NATO ASI Series (Series F: Computer and Systems Sciences), vol. 30, Springer, Berlin, Heidelberg, 1987.

[2] W. Bożejko, M. Uchroński, and M. Wodecki: Parallel metaheuristics for the cyclic flow shop scheduling problem, *Computers & Industrial Engineering*, **95** (2016), 156–163.

[3] W. Bożejko and M. Wodecki: Solving Permutational Routing Problems by Population-Based Metaheuristics, *Computers & Industrial Engineering*, **57** (2009), 269–276.

[4] W. Bożejko and M. Wodecki: Parallel Evolutionary Algorithm for the Traveling Salesman Problem, *Journal of Numerical Analysis, Industrial and Applied Mathematics*, **2**(3-4) (2007), 129–137.

[5] Civil Aviation Office, Report – number of valid licenses for day 31.12.2017. (2018) [online] [Accessed 1 Jul. 2019].

[6] R.H. Byrd, P. Lu, and J. Nocedal: A limited memory algorithm for bound constrained optimization, *SIAM Journal on Scientific Computing*, **16**(5) (1995), 1190–1208.

[7] G.A. Croes: A method for solving traveling-salesman problems, *Operations Research*, **6**(6) (1958), 791–812.

[8] J. Holden and N. Goel: *Fast-forwarding to a future of on-demand urban air transportation*, San Francisco, CA, 2016.

[9] F. Imeson and S.L. Smith: Multi-robot task planning and sequencing using the SAT-TSP language, *IEEE International Conference on Robotics and Automation* (2015), 5397–5402.

[10] D. Kraft: *A software package for sequential quadratic programming*, Forschungsbericht, Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt, 1988.

[11] N. Mathew, S.L. Smith, and S.L. Waslander: Multirobot rendez vous planning for recharging in persistent tasks, *IEEE Trans. Robot.*, **31**(1) (2015), 128–142.

[12] J.A. Nelder, and R. Mead: A simplex method for function minimization, *The Computer Journal*, **7**(4) (1965), 308–313.

[13] J. Nocedal and S. Wright: *Numerical optimization*, Springer Science & Business Media (2006), 529–562.

[14] M.J.D. Powell: An efficient method for finding the minimum of a function of several variables without calculating derivatives, **The Computer Journal**, **7**(2) (1964), 155—162.

[15] L. Snyder and M. Daskin: A random-key genetic algorithm for the generalized traveling salesman problem, *European Journal of Operational Research*, **17**(1) (2006), 38–53.

[16] A. Stoschek: Exploring Sense-and-Avoid Systems for Autonomous Vehicles, Vahana, [Accessed November 1, 2020].

[17] M. Wodecki and W. Bożejko: Solving the flow shop problem by parallel simulated annealing, Lecture Notes in Computer Science No. 2328, Springer (2002), 236–247.

[18] E.M. Wolff, U. Topcu, and R.M. Murray: Optimal control of non-deterministic systems for a computationally efficient fragment of temporal logic. *IEEE Conference on Decision and Control* (2013), 3197–3204.

[19] C. Zhu, R.H. Byrd, P. Lu, and J. Nocedal: Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization, *ACM Transactions on Mathematical Software* (*TOMS*), **23**(4) (1997), 550–560.