An Evolutionary Algorithm for Online, Resource-Constrained, Multivehicle Sensing Mission Planning

Nikolaos Tsiogkas [ID] and David M. Lane

Abstract—Mobile robotic platforms are an indispensable tool for various scientific and industrial applications. Robots are used to undertake missions whose execution is constrained by various factors, such as the allocated time or their remaining energy. Existing solutions for resource constrained multirobot sensing mission planning provide optimal plans at a prohibitive computational complexity for online application. A heuristic approach exists for an online, resource-constrained sensing mission planning for a single vehicle. This letter proposes a *genetic algorithm* (GA) based heuristic for the *correlated team orienteering problem* that is used for planning sensing and monitoring missions for robotic teams that operate under resource constraints. The heuristic is compared against optimal *mixed integer quadratic programming* solutions. Results show that the quality of the heuristic solution is at the worst case equal to the 5% optimal solution. The heuristic solution proves to be at least 300 times more time efficient in the worst tested case. The GA heuristic execution required in the worst case less than a second making it suitable for online execution.

Index Terms—Multi-robot systems, path planning for multiple mobile robots or agents, planning, scheduling and coordination.

I. INTRODUCTION

ROBOTIC technology has significantly advanced during the past years. The maturing process of the scientific breakthroughs, combined with engineering effort, has led to reliable and affordable systems, that are becoming an indispensable assistant to science and industry. Some examples include space exploration [5], underwater archaeology [6] and search and rescue [7].

In many cases the robot operation is constrained by the nature of the problem the robot is solving. For example, in emergency situations time is of essence. It is required to gather the maximum amount of information in a limited amount of time so that a response plan can be devised. In other cases the dynamic nature of the environment affects the energy requirements, leaving the vehicle with less energy to complete the required mission. For example, in a highly tidal environment the developed currents are strong and have an effect on the mission execution energy and time [8]. In such cases it is required to maximise the

sensing outcome with limited energy by being able to replan a maximising trajectory online.

This letter focuses on sensing missions where specific resource constraints need to be respected. The mission requires a team of vehicles to sample a user defined area for scalar field estimation. A scalar field is used to represent the physical quantities in space. Its importance is presented in [9], where phenomena such as algal blooms, oil spills and chemical pollution are studied. These phenomena are dynamic and they need to be monitored fast and precisely in order to be tackled. An example can be seen in [10] with the use of Autonomous Underwater vehicles for monitoring in the Deepwater Horizon oil spill. For these cases [1], [2] presented a variant of the *Orienteering Problem* (OP) for persistent monitoring tasks using *Unmanned Aerial Vehicles* (UAVs). The presented approach takes into account the correlation of information among the sampling points, while respecting some budget constraints. The problem was named the *Correlated Orienteering Problem* (COP) and *Mixed Integer Quadratic Programming* (MIQP) formulations were provided for a single (COP) and multiple vehicles (CTOP). The method was able to generate paths for one or more vehicles that were optimal for the monitoring task.

In [3] the COP method was adapted for an underwater sensing scenario. Results showed its potential over the standard OP for field estimation, but its computational complexity made it prohibitive for online use. To overcome that, in [4], a GA based heuristic was presented. The presented heuristic was able to generate paths for a single vehicle that were close to optimal, with low computational cost, making it suitable to be used online.

The work presented in this letter is focused on heuristic solutions for the multi-vehicle case of the COP. It provides a GA based heuristic method that aims to generate close to optimal paths for the whole team, while keeping the time complexity low. This would allow plans to be regenerated online in case of a change in the mission parameters. For example, if some constraint changes, or in case of a vehicle failure, a new plan can be generated for the team, guaranteeing a close to optimal execution any time.

The rest of the paper is organised as follows. Section II presents a brief review on previous work in the field of resource constrained optimisation. In Section III the CTOP and the proposed heuristic solution are detailed. Section IV presents the results obtained through simulated experiments. Finally, in Section V the paper concludes discussing potential future paths for research.

II. Previous Work

In the literature various approaches to maximise the utility gathered from an agent that operates under some resource constraint can be found. A formal definition of the problem was first presented in [11] as the selective travelling salesman problem. The work of [12] coined the term *orienteering problem* (OP), as the same problem describes the sport of orienteering. Applications of the problem to transportation and logistics systems are presented in [13], [14].

Extending the classic OP to multiple agents is mentioned in the literature as the *team orienteering problem* (TOP). Over the years multiple heuristic solutions have been presented. In [15] a tabu search heuristic is used. Local search methods are presented in [16] and [17]. The work of [18] presents an ACO approach, while [19] uses simulated annealing to solve the problem. Variable neighbourhood search is used in [20]. In [21] *variable neighbourhood search* (VNS) is used to tackle the Capacitated TOP with time windows. A path relinking approach for the TOP is presented in [22]. In [23] the TOP with time windows is solved using an evolutionary local search algorithm.

The tasks of sampling or information gathering have been modelled as instances of the OP in various works. In [24] it is applied to lake and river monitoring. Approximation methods of a submodular OP are used to maximise the information gathering using autonomous surface vehicles. In [25] a recursive greedy algorithm is used to solve the additional information gathering problem in a sensor network. The work of [26] found the submodular OP inapplicable in real-time tasks. It proposes the usage of a linear approximation of the submodular OP for area coverage using a micro-UAV. A branch-and-bound solution to the OP is used in [27] for scalar field estimation using Gaussian processes. In [28] a GA based heuristic is used to solve an OP for marine sampling. In [29] an agricultural monitoring application is shown using multiple robots for sensing. The work of [30] presents a vehicle routing problem with time windows variant that can't serve all the customers so it is forced to select customers based on revenue. That problem formulation is directly related with the TOP. The problem is solved using a branch-and-price approach. In [31] an evolutionary strategy is used for online terrain classification using unmanned aerial vehicles. Informative path planning under disturbances is studied in the work of [32]. They provide a dynamic programming solution to the addressed path planning problem. In [33], [34] they present a VNS approach for Dubins vehicles to the OP and COP respectively. Their approach successfully computes optimised paths, but the computational cost is prohibitive for online application. The work of [35] provides a randomised algorithm for the informative path planning for a single vehicle. The computational time is allowing it to be applied online. To the best of our knowledge there is no prior work to tackle the online correlated informative path planning under budget constraints for multiple vehicles.

III. Methods

In this section the proposed solution and the methods that it consists of are presented in detail. Section III-A briefly presents a method for scalar field estimation using Gaussian processes. In Section III-B the MIQP formulation of the CTOP is presented.

Section III-C presents the details of the proposed evolutionary heuristic.

A. Scalar Field Estimation

As mentioned in Section I the mission requires a scalar field to be estimated. A method to model such a field based on a *Gaussian process* (GP) is presented in [27]. To achieve that, a grid of sampling points is used to approximate the field. The field values are modelled as a multivariate Gaussian distribution, where one dimension is used for each sampling point. The covariance is calculated using a kernel function, such as the square exponential kernel shown in (1).

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|}{2l^2}\right) \qquad (1)$$

The kernel is used to model the drop in correlation of two points based on their distance. The parameter l governs that behaviour and can be estimated from previously gathered data. In general, the distance in which two points can be correlated is larger, when l grows larger. This is the correlation taken into account in the aforementioned CTOP formulation.

B. Correlated Team Orienteering Problem

The CTOP, which was first defined in [1], [2], maximises the following objective function:

$$\sum_{i \in V} \left(r_i x_i + \sum_{v_j \in N_i} r_j w_{ij} x_i (x_i - x_j) \right) \qquad (2)$$

For any given solution the objective function represents the sum of the reward of each of the visited vertices in that solution. All the vertices are included in set V. The variable r_i represents the reward for visiting vertex i in a solution. To encode that a vertex i is visited in a given solution, the binary variable x_i is used. In addition to the reward received for visiting a vertex i, extra reward is obtained from its neighbourhood N_i. This reward is calculated by summing the reward of each unvisited neighbour j multiplied by a weight w_{ij}. To ensure that only reward from unvisited vertices will be added, the quadratic term $x_i(x_i - x_j)$ is used.

To enforce that m vehicles start and finish in user defined points, constraints (3) and (4) are used.

$$\sum_{i \in V \setminus \{s\}} x_{isk} = \sum_{i \in V \setminus \{f\}} x_{fik} = 0, \qquad \forall 1 \le k \le m \qquad (3)$$

$$\sum_{i \in V \setminus \{s\}} x_{sik} = \sum_{i \in V \setminus \{f\}} x_{ifk} = m, \qquad \forall 1 \le k \le m \qquad (4)$$

In this formulation a binary variable x_{ijk} is showing that a path exists from vertex i to vertex j in the path of vehicle k. Specifically, x_{sik} shows that any vertex i is visited right after the starting vertex s for the path of vehicle k. Likewise, variable x_{ifk} defines that the finishing vertex is after vertex i. For all the other vertices in the solution constraints (5)–(9) are applied.

$$\sum_{j \in V \setminus \{sf\}} x_{ijk} \le 1, \qquad \forall i \in V \setminus s, f, 1 \le k \le m \qquad (5)$$

$$\sum_{j \in V \setminus \{sf\}} x_{jik} \le 1, \qquad \forall i \in V \setminus s, f, 1 \le k \le m \quad (6)$$

In the solution, each vertex is visited at most once, as it is ensured from constraints (5) and (6). This behaviour is enforced by allowing at most one path entering and one leaving in each of the vertices in V, other than the start and finish. This happens for all the m paths.

$$\sum_{k=1}^{m} \sum_{j \in V \setminus \{sf\}} x_{ijk} = x_i \le 1, \qquad \forall i \in V \setminus f \quad (7)$$

$$\sum_{k=1}^{m} \sum_{j \in V \setminus \{sf\}} x_{jik} = x_i \le 1, \qquad \forall i \in V \setminus s \quad (8)$$

$$\sum_{j \in V \setminus \{sf\}} x_{ijk} = \sum_{j \in V \setminus \{sf\}} x_{jik}, \qquad \forall i \in V, 1 \le k \le m \quad (9)$$

Constraints (7) and (8) make sure that if a vertex j is visited in a solution, a path that connects j with a next vertex exists. Constraint (9) prevents a vertex having a path from one vehicle entering and the path of a different vehicle exiting it. The maximum amount of resources used is enforced by constraint (10).

$$\sum_{i \in V} \sum_{j \in V} (c_{ij} + c_i) x_{ijk} \le C_k, \qquad \forall 1 \le k \le m \quad (10)$$

Term c_i represents a fixed cost of performing sensing in vertex i. To that it is added the cost of travelling from vertex i to the next vertex c_{ij}. Then the cost is multiplied by the binary variable x_{ijk}, used to denote the existence of a path between i and j in path of robot k. The sum of all these costs must be less or equal to the maximum allowed resource usage for robot k. It must be noted that the fixed cost of operations of the user defined start and finish vertices is zero.

$$u_{ik} - u_{jk} + 1 \le (|V| - 1)(1 - x_{ijk})$$
$$\forall i, j \in V, i \ne j, \forall 1 \le k \le m \quad (11)$$
$$0 \le u_{ik} \le |V|$$
$$\forall i \in V, \forall 1 \le k \le m \quad (12)$$

Finally, constraints (11) and (12) are subtour elimination constraints, allowing only a single tour to be generated for each robot.

C. Genetic Algorithm Heuristic

Genetic algorithms are one of the main meta-heuristics used for optimisation. They try to mimic the evolution process occurring in nature by using a population of chromosomes that undergo processes of natural selection, reproduction and mutation as described in [36]. The genetic algorithm presented by this letter uses the chromosome encoding as presented in [37]. It is composed by a set vectors, or genes, each one of which is representing a path for a single robot. In each vector the indices of the vertices to be visited by that robot is stored. The whole chromosome, therefore, represents a candidate CTOP solution. An outline of the procedure used during the optimisation can be seen in algorithm 1. It can be seen that the optimisation

---

Algorithm 1: Genetic Algorithm for the Correlated Team Orienteering Problem.

Input: $TravelCosts, Rewards, MaxCosts, Start, Finish$
Output: $FittestChromosome$
1: INITIALISEPOPULATION
2: **while** $gen \le MaxGen$ or $FittestChromosome$ stable for 10 generations **do**
3:     SELECTNEWPOPULATION
4:     CROSSOVER
5:     MUTATE
6: **select** $FittestChromosome$
7: **return** $FittestChromosome$

Algorithm 2: Random Gene Generation.

Input: $TravelCosts, Rewards, MaxCost, Start, Finish, AvailVertices$
Output: $Gene, AvailVertices$
1: INSERT($Gene, Start$)
2: $Done \leftarrow False$
3: **while** $!Done$ **and** $AvailVertices \ne \varnothing$ **do**
4:     $v \leftarrow$ SELECTRANDOM($AvailVertices$)
5:     **if** COSTINSERT($Gene, v$) $\le MaxCost$ **then**
6:         INSERT($Gene, v$)
7:         $AvailVertices \leftarrow AvailVertices \setminus v$
8:     **else**
9:         $Done \leftarrow True$
10: INSERT($Gene, Finish$)
11: **return** $Gene, AvailVertices$

---

procedure is separated in two distinct phases. The first phase is responsible for the population generation, while the second phase repeatedly applies the operations of selection, crossover and mutation to optimise the population.

In the scope of this letter two different chromosome generation methods are presented and compared. The difference is based on the way the genes, composing the chromosome, are generated. The first method is a random method, similar to the one presented in [4]. It can be seen in algorithm 2.

In the random generation method genes are created by randomly inserting vertices until the maximum cost is reached or until there are no more free vertices to insert. To generate a chromosome this process is repeated a number of times, equal to the number of robots. In the chromosome generation process, the set of available vertices is shared among the genes, so that if a vertex is picked up by a robot, it cannot be assigned to any other robot.

The second gene generation method is named *Nearest Neighbour Randomised Adaptive Search Procedure* (NN-RASP). It is inspired by the *Greedy Randomised Adaptive Search Procedure* (GRASP) that is used in [22] to provide a heuristic solution to the TOP. This procedure is taking into consideration the peculiarities of the CTOP problem, regarding the correlation of information. It and can be seen in algorithm 3.

This method is based on progressively constructing a path by choosing one of the neighbours of the last inserted vertex

Algorithm 3: Nearest Neighbour Randomised Adaptive Search Procedure.

Input: $TravelCosts, Rewards, MaxCost, Start, Finish$
Output: $Gene$
1: INSERT($Gene, Start$)
2: $v \leftarrow$ GETLAST($Gene$)
3: $N \leftarrow$ GETNEIGHBOURS(v)
4: **while** GETSIZE(N) > 0 **do**
5:     **for each** $n \in N$ **do**
6:         GETDISTANCEWEIGHT(n)
7:         GETNEIGHBOURWEIGHT(n)
8:         GETCOMBINEDWEIGHT(n)
9:     $nv \leftarrow$ GETWEIGHTEDRANDOM(N)
10:     **if** COSTINSERT($Gene, nv$) $<=$ $MaxCost$ **then**
11:         INSERT($Gene, nv$)
12:         $v \leftarrow$ GETLAST($Gene$)
13:         $N \leftarrow$ GETNEIGHBOURS(v)
14:     **else**
15:         $N \leftarrow \{\varnothing\}$
16: INSERT($Gene, Finish$)
17: **return** $Gene$

Algorithm 4: Chromosome Evaluation.

Input: $TravelCosts, Rewards, MaxCost$
Output: $Fitness$
1: **for each** $g \in Genes$ **do**
2:     TWOOPT(g)
3:     REMOVEDUPLICATES(g)
4:     $GeneFit \leftarrow$ GETREWARD(g)^3/GETCOST(g)
5:     $Fitness \leftarrow Fitness + GeneFit$
6: **return** $Fitness$

Algorithm 5: Chromosome Crossover.

Input: $Parent_1, Parent_2$
Output: $Child_1, Child_2$
1: SORTGENES($Parent_1$)
2: SORTGENES($Parent_2$)
3: **for each** $i \in 1, 2$ **do**
4:     $tP_1 \leftarrow Parent_1$
5:     $tP_2 \leftarrow Parent_2$
6:     **while** GETNUMGENES($Child_i$) < $numRobots$ **do**
7:         $p \leftarrow$ SELECTRANDOMPARENT(tP_1, tP_2)
8:         INSERTBESTGENE($Child_i, p$)
9:         REMOVEVERTICES (tP_1, tP_2)
10:         EVALUATEANDSORTGENES tP_1, tP_2
11:     EVALUATECHROMOSOME($Child_i$)
12: **return** $Child_1, Child_2$

in the path. The neighbour to be inserted is chosen based on a categorical distribution where each one has a probability to be chosen. This probability is based on the distance of the neighbouring vertex from all other visited vertices in the solution and the number of free neighbours it has. The distance from other vertices is chosen as this can reduce overlapping of paths. The number of free neighbours is used as it gives a better reward from correlated information, as well as, it will allow the path to have more options to continue its construction. When a vertex is chosen it is attempted to be inserted in the gene. If the gene that is resulting is feasible the vertex is inserted and the search continues, else the search stops and the gene is returned. The chromosome generation is again performed by iteratively calling the gene generation methods until paths are generated for all the robots.

To create the population both chromosome generation procedures are repeated until the specified amount of chromosomes is generated. Following their generation, the chromosomes are evaluated according to algorithm 4.

The fitness of each chromosome is defined as the sum of fitnesses of each gene composing it. Each gene goes through a *2-Opt* operation to minimise its cost. This operation is a simple local search method that is based on swapping segments of a path in hope of reducing the path's cost. It is iteratively applied to all segment combinations and stops if no improvement can be

achieved. It was first presented in [38] as a method to solve the travelling salesman problem. Then any duplicate vertices that have been already visited by other genes are removed. Its fitness is calculated as the total reward to the third power divided by the gene cost as suggested in [39] for the single agent orienteering problem.

After the population generation is completed, a repetitive process is applied. This process is applied until either the best chromosome is stable for 10 generations or a maximum amount of generations is reached. This process involves selecting a new population based on the previous one. A percentage of the old population is passed to the next generation without selection. This process is described in the literature as *Elitism* and is known to speed up the performance and prevent loss of good solutions during the search process [40]. The rest of the population is selected using *Tournament selection*. This method starts with an empty population, then repetitively chooses n random individuals from the previous population and adds the fittest to the new population, until the maximum amount of individuals is chosen.

Following the new population selection a crossover operation is performed to a percentage of the population. The operation is presented in algorithm 5 and is a modified version of the one presented in [37]. The crossover operation aims to diversify the population and explore more of the search space.

For this operation to take place, two chromosomes are randomly chosen as the parents of the crossover and they are replaced by the two produced offsprings. Initially the genes of each parent are sorted by fitness. Then, the two children are constructed iteratively. The first step of the iteration involves copies of the parents to be created. Next, as long as the number of genes in each child is less than the number of robots, the following process takes place. A parent is selected randomly and its best gene is removed and inserted to the child. Then, the vertices of this gene are removed from any other genes of the temporary parents. After that the genes of the parents are evaluated again and sorted based on their fitness. When the maximum amount of genes is inserted in the child, it is accordingly evaluated. After both offsprings are generated, they are returned in the place of their parents.

Continuing the population crossover, a percentage of the population is evolving by mutating. This aims at attempting to

---

Algorithm 6: Chromosome Mutation.

Input: $Chromosome, MaxCost, NumMut, AddProb$
Output: $Chromosome$
1: $Genes \leftarrow$ GETGENES($Chromosome$)
2: **for each** $g \in Genes$ **do**
3:    **for** $i \leftarrow 1, NumMut$ **do**
4:       $p \leftarrow$ GETRANDOM(0, 1)
5:       **if** $p \leq AddProb$ **then**
6:          **if** GETCOST(g) $\geq 0.95 * MaxCost$ **then**
7:             $v \leftarrow$ SELECTRANDOM($g, 1$)
8:             $n \leftarrow$ GETMAXFREENEIGHBOUR(v)
9:             $gn \leftarrow$ REPLACE(g, v, n)
10:           **if** FITNESS(g) $\leq$ FITNESS(gn) **then**
11:             $g \leftarrow gn$
12:          **else**
13:           **if** $FV \neq [\varnothing]$ **then**
14:             $v \leftarrow$ SELECTRANDOM($FV, 1$)
15:             $idx \leftarrow$ FINDBESTINSERTION(g, v)
16:             INSERT(g, v, idx)
17:       **else**
18:          $v \leftarrow$ GETMINLOSSVERTEX(g)
19:          $g \leftarrow$ REMOVE(g, v)
20: EVALUATECHROMOSOME($Chromosome$)
21: **return** $Chromosome$

---

improve the fitness of each gene that composes the mutating chromosome and thus improve the fitness of the chromosome itself. The mutation process resembles the one presented in [4] and can be seen in algorithm 6.

For each gene in the chromosome a total number of $NumMut$ mutations will take place. In the case studied in this work $NumMut$ was empirically set to 10 due to good performance in [4]. A mutation can happen in two ways, by either adding a vertex in the gene, or by removing one. This is governed by the $AddProb$ probability and was set to be 0.9. In the first case there are two behaviours depending on the cost of the gene that is mutated. If the cost is 95% of the maximum cost, an improvement of the solution is attempted by swapping a vertex of the solution with one of its free neighbours. A random vertex in the solution is chosen and is iteratively swapped with all its free neighbours. The swap with the maximum improvement in the gene's fitness is kept as the performed mutation. In the second behaviour, a free vertex is randomly chosen. Then the best insertion place is found in the path. It is consequently inserted if it does not violate the maximum cost constraint. The best insertion is found using the heuristic value described in the GRASP method in [22]. In the second case, where a removal is performed, the minimum loss vertex for this gene's fitness is found and removed. Likewise with the insertion, the minimum loss vertex is found using the same heuristic value. After all the $NumMut$ mutations happen for all the genes, the chromosome is evaluated using the method already described and is returned.

IV. EXPERIMENTAL RESULTS

This section describes the experimental results obtained through simulations. The planning is required for a simulated sampling mission that is to be performed by a team of vehicles.

The mission is composed by a set of *sampling points*, arranged in a grid. The team is given specific starting and finishing points. Additionally, each vehicle is allowed to use a limited amount of budget to complete the mission. The overall goal of the team is to maximise the utility obtained without exceeding the budget limitations.

The experiments are divided into two sections. In the first section a method from the literature is used to tune the different parameters that govern the performance of the heuristic algorithm. In the second section the tuned parameters are used and proposed heuristic is compared against the optimal MIQP solution. The two methods are compared based on the utility obtained by the team and the time it took to generate a plan. The grid was chosen to be 9 by 9, as choosing a larger would make it impractical to collect results with the exact method. To further limit the planning time of the exact MIQP method, it was stopped once the solution found was less than 5% different from the optimal. These results are referred with the term $CTOP5\%$. For the heuristic, given that it is a random process, 1000 experiments were run for each case and statistical results are presented.

All the experiments were run on a system having an Intel i5-7600 CPU, 16 GB of RAM and running on Ubuntu 16.04. The MIQP instances were solved using the Gurobi 7.0.2 commercial solver [41]. The heuristic was coded in C++11 and compiled using g++ version 5.4.0. Source code for both approaches can be found online[1],[2].

A. Algorithm Tuning

As described in Section III, a GA uses a set of parameters to manage its behaviour. A tuning process is essential to find the best parameters for the algorithm. In the literature one can find methods for parameter tuning. In [42] the general framework for parameter tuning is presented and various methods are discussed. In the scope of this work a modified version of the work presented in [43] was used.

The parameter tuning method that [43] presents is based on an evolutionary approach, using chess rating for evaluation. Each chromosome in this method is represented by a vector of integer and real values representing the set of parameters. To ensure that the tuned parameters are generalising well, each parameter set is tested against a set of problems. For tuning the presented GA, twelve problem instances were created. They were created by altering the problem size, the vertex distribution, and the number of vehicles. Problem instances of $5 \times 5, 7 \times 7$ and 9×9 were used. The vertices were distributed in a grid and a noisy grid form. The noisy grid was created based on the canonical grid and adding some uniform random noise to the position of each vertex. The number of vehicles were 3 and 5. The budget was set to 75% of the maximum budget for all the problems. The rest of the approach is presented in algorithm 7.

The first step of the tuning algorithm is to generate a population of chromosomes. Each chromosome is constructed by randomly choosing values for the parameters from a specified range. For the tuning procedure performed for this work, the chromosome population was composed by 100 individuals.

---

[1]https://github.com/lounick/lwga
[2]https://github.com/lounick/task_scheduling

Algorithm 7: CRS-Tuning algorithm.

Input: *Problems*
Output: *BestConfiguration*
1: $C \leftarrow$ INITIALISEPOPULATION
2: **for** $trial \leftarrow 1, MaxTrials$ **do**
3:     **for each** $c \in C$ **do**
4:         **for each** $pinProblems$ **do**
5:             **for (do** $g \in 1, NumGames$**) do**
6:                 $s \leftarrow GA(c, p)$
7:                 $S_{c,p,g} \leftarrow s$
8:     EVALUATECONFIGURATIONS (C, S)
9:     $parents \leftarrow$ SELECTPARENTS(C)
10:    $newC \leftarrow$ GENERATENEWPOP$(parents)$
11:    $C \leftarrow newC$
12: **select** *BestConfiguration*
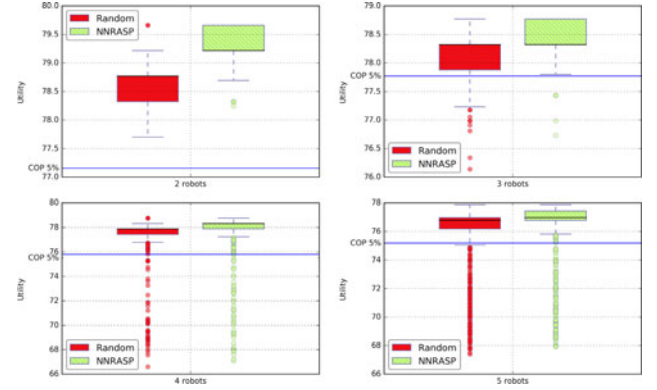13: **return** *BestConfiguration*

Fig. 1. Utility box plots for different numbers of robots. For lower number of robots a sophisticated population initialisation method tends to give better results. As the number goes higher the results come closer with a slight superiority of the NNRASP method. The blue line represents the 5% gap optimal solution obtained by solving the MIQP problem.

TABLE I
PARAMETER RANGES AND TUNED VALUES FOR THE TWO DIFFERENT METHODS

Parameter	Value Range	Random	NNRASP
Population Size	$[25, 50, \ldots, 500]$	300	250
Generations Number	$[5, 10, \ldots, 50]$	40	50
Tournament Size	$[3, 4, \ldots, 10]$	6	5
CX Probability	$[0.0, 0.1, \ldots, 0.9]$	0.7	0.9
Mutation Probability	$[0.0, 0.1, \ldots, 0.9]$	0.6	0.7
Elitist Percentage	$[0.01, 0.02, \ldots, 0.20]$	0.19	0.03

TABLE II
AVERAGE TIME(S) FOR DIFFERENT ALGORITHMS AND VEHICLE NUMBERS

Algorithm	Number of Robots			
	2	3	4	5
CTOP 5%	396.36	695.60	2830.41	1474.18
GA-Random	**0.787703**	**0.729135**	**0.666236**	**0.623074**
St.Dev.	0.087695	0.0738167	0.0602966	0.0579891
GA-NNRASP	0.792475	0.844305	0.817855	0.826688
St.Dev.	0.106818	0.120596	0.112062	0.106136

Then the algorithm iteratively runs for *MaxTrials*. For this work *MaxTrials* were empirically set to ten. Initially, each configuration is used to run each problem *NumGames* times and their scores are recorded. Here *NumGames* was empirically set to ten as well.

Following, the configurations are evaluated using the *Glicko2* chess rating system [44]. The evaluation is performed by each configuration comparing its results against all the other configurations results and collecting 0, 0.5 and 1 points for loss, draw and win respectively. These points, as well as the opponents rating, are used to update the fitness of each configuration. In [43] the evaluation is performed based only on the scores of the games. For the tuning performed in this letter a bonus of 0.1 was given to the faster solution, while the slower solution got a penalty of -0.1. This was done to differentiate the cases where two solutions were equally good utility-wise but one was slower to compute than the other.

The next step involved selecting the parents that would generate the next generation of configurations. The ten best configurations were selected as parents as well as any other parents that performed close to the first ten. The choice was performed based on their chess rating intervals. The parents had a maximum size of half the total population. These new parents were a portion of the new population. The rest of the population was generated by applying uniform crossover and mutation operations with 50% and 80% percent respectively, as suggested by [43]. Finally the best configuration was selected and returned.

The tuning method was run for both population generation methods presented in Section III. The resulted configurations as well as the parameter ranges can be seen in Table I.

B. Results

The performance of the heuristic method is compared against the optimal solution in two different ways. The first way examines how well the heuristic method performs for different numbers of robots. For that the maximum budget was allocated to each robot. The maximum budget was calculated by taking the maximum budget required for a single robot to complete the mission and dividing it by the number of robots. The results can be seen in Fig. 1 and Table II.

Fig. 1 presents statistical results regarding the utility for each size of the robotic team using the GA based planning. It is compared against a solution produced by solving the MIQP problem using the Gurobi solver. As it was already mentioned, the solver was stopped once the found solution was guaranteed to be at most 5% worse than optimal. This solution is represented by the horizontal blue line. It can be seen that the GA heuristic with the NNRASP generation method outperforms the MIQP solution in all the cases. The random generation method falls a bit behind, with part of the results produced being worse than the 5% optimal.

Comparing the average planning time the GA heuristic heavily outperforms the MIQP exact method, as it can be seen in Table II. It can be seen that the time required for the optimal solution is at the best case 500 times higher. The GA-Random method performed slightly better than the GA-NNRASP. Example paths for a three robot team can be seen in Fig. 2(a)–(c). The first figure shows the 1% optimal path calculated by the MIQP solver, while the second is a 5% optimal solution used
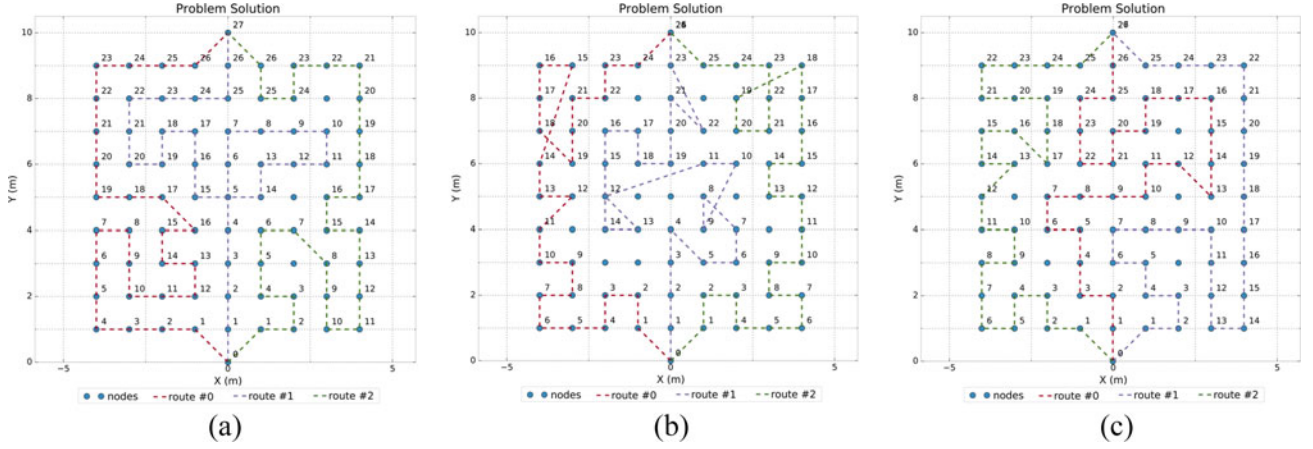
Fig. 2. Comparison of the paths generated for three vehicles having full budget. (a) 1% optimal MIQP solution for three vehicles. The gathered utility is 79.662 and the time to calculate is 5847.81 seconds. (b) 5% optimal MIQP solution for three vehicles. The gathered utility is 76.433 and the time to calculate is 690.74 seconds. (c) GA heuristic solution for three vehicles. The gathered utility is 78.770 and the time to calculate is 0.8597 seconds.
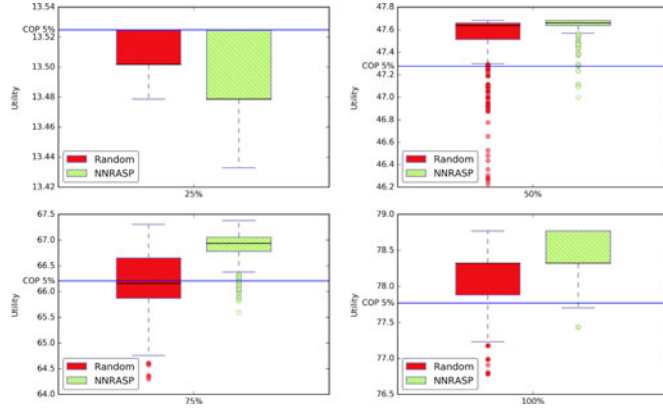

Fig. 3. Utility box plots for three robots having different budget. For lower budgets a sophisticated population initialisation performs equally good with the random one. As the budget goes higher the results show the benefits of the NNRASP method. The blue line represents the 5% gap optimal solution obtained by solving the MIQP problem.

TABLE III
AVERAGE TIME(S) FOR DIFFERENT ALGORITHMS AND DIFFERENT BUDGETS

Algorithm	Budget			
	100%	75%	50%	25%
CTOP 5%	695.60	267.94	3611.76	3764.12
GA-RANDOM	**0.721606**	**0.625391**	**0.362355**	**0.141678**
St.Dev.	0.0742465	0.0607859	0.0346406	0.0147053
GA-NNRASP	0.837791	0.763285	0.389283	0.165087
St.Dev.	0.120398	0.102063	0.0483728	0.0244343

for comparison with the GA. The heuristic solution can be seen in the third image. It is obvious that the solution quality of the GA is far better than the 5% optimal.

The second performance metric examines how the heuristic method performs in cases where the budget is limited. For this the number of vehicles is chosen to be three and is kept static. Then the budget of each vehicle is reduced by a specific percentage. Fig. 3 and Table III present the results for this metric.

In Fig. 3 the utility gained for each budget is shown. It can be seen that the GA method performs at least equally good as the 5% exact method. That happened only in the case where the robots had 25% of the maximum energy. In all the other cases the GA performed better. Comparing the two different population generation methods, one can see that the more sophisticated approach gives statistically better results. The random method performs close to the 5% optimal.

Table III the timing results of the limited budget experiments are presented. As with all the previous cases the GA-Random method performed better. In the worst case it was more than 300 times faster than the 5% MIQP solution. The GA with NNRASP

population generation method performed constantly a bit slower than the Random generation method.

V. CONCLUSION

This letter studies the online task scheduling problem for sensing missions performed by multiple vehicles. Such problems can be solved using the correlated team orienteering problem. The CTOP is designed for such problems as it takes into account the correlation of sensed information between various sensing points, while keeping the resource usage constrained. This work presents a heuristic for solving the CTOP that is based on a genetic algorithm. Initially, it uses a method from the literature to tune the GA parameters. The heuristic is then compared against a mixed integer quadratic programming solution. It is shown that in terms of utility the heuristic performs usually better than a near optimal MIQP solution. In terms of solution time complexity the heuristic performs at least 300 times better than the MIQP. The average execution time for the studied instances was in the worst case less than a second. These results make the proposed GA heuristic suitable for online usage. Future work would involve testing the heuristic in larger, more complex instances where the exact solution is inapplicable. Finally, it would be worth exploring the coordination capabilities of a heterogeneous team with different capabilities and objectives using a solution produced by a single instance of the proposed heuristic.

References

[1] J. Yu, M. Schwager, and D. Rus, "Correlated orienteering problem and its application to informative path planning for persistent monitoring tasks," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2014, pp. 342–349.

[2] J. Yu, M. Schwager, and D. Rus, "Correlated orienteering problem and its application to persistent monitoring tasks," *IEEE Trans. Robot.*, vol. 32, no. 5, pp. 1106–1118, Oct. 2016.

[3] N. Tsiogkas, V. De Carolis, and D. M. Lane, "Energy-constrained informative routing for auvs," in *Proc. OCEANS 2016-Shanghai*, 2016, pp. 1–5.

[4] N. Tsiogkas, V. De Carolis, and D. Lane, "Towards an online heuristic method for energy-constrained underwater sensing mission planning," in *Proc. IEEE RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 6801–6808.

[5] W. F. Truszkowski, M. G. Hinchey, J. L. Rash, and C. A. Rouff, "Autonomous and autonomic systems: A paradigm for future space exploration missions," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 36, no. 3, pp. 279–291, May 2006.

[6] B. Allotta *et al.*, "The arrows project: Adapting and developing robotics technologies for underwater archaeology," *IFAC-PapersOnLine*, vol. 48, no. 2, pp. 194–199, 2015.

[7] T. Tomic *et al.*, "Toward a fully autonomous UAV: Research platform for indoor and outdoor urban search and rescue," *IEEE Robot. Autom. Mag.*, vol. 19, no. 3, pp. 46–56, Sep. 2012.

[8] J. Bellingham and J. Willcox, "Optimizing AUV oceanographic surveys," in *Proc. Symp. Auton. Underwater Veh. Technol.*, Jun. 1996, pp. 391–398.

[9] H. M. La, W. Sheng, and J. Chen, "Cooperative and active sensing in mobile sensor networks for scalar field mapping," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 1, pp. 1–12, Jan. 2015.

[10] J. C. Kinsey, D. R. Yoerger, M. V. Jakuba, R. Camilli, C. R. Fisher, and C. R. German, "Assessing the deepwater horizon oil spill with the sentry autonomous underwater vehicle," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 261–267.

[11] G. Laporte and S. Martello, "The selective travelling salesman problem," *Discrete Appl. Math.*, vol. 26, no. 2/3, pp. 193–207, 1990.

[12] I.-M. Chao, B. L. Golden, and E. A. Wasil, "A fast and effective heuristic for the orienteering problem," *Eur. J. Oper. Res.*, vol. 88, no. 3, pp. 475–489, 1996.

[13] P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden, "The orienteering problem: A survey," *Eur. J. Oper. Res.*, vol. 209, no. 1, pp. 1–10, 2011.

[14] A. Gunawan, H. C. Lau, and P. Vansteenwegen, "Orienteering problem: A survey of recent variants, solution approaches and applications," *Eur. J. Oper. Res.*, vol. 255, no. 2, pp. 315–332, 2016.

[15] H. Tang and E. Miller-Hooks, "A tabu search heuristic for the team orienteering problem," *Comput. Operations Res.*, vol. 32, no. 6, pp. 1379–1407, 2005.

[16] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. Van Oudheusden, "Iterated local search for the team orienteering problem with time windows," *Comput. Oper. Res.*, vol. 36, no. 12, pp. 3281–3290, 2009.

[17] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. Van Oudheusden, "A guided local search metaheuristic for the team orienteering problem," *Eur. J. Oper. Res.*, vol. 196, no. 1, pp. 118–127, 2009.

[18] L. Ke, C. Archetti, and Z. Feng, "Ants can solve the team orienteering problem," *Comput. Ind. Eng.*, vol. 54, no. 3, pp. 648–665, 2008.

[19] S.-W. Lin and F. Y. Vincent, "A simulated annealing heuristic for the team orienteering problem with time windows," *Eur. J. Oper. Res.*, vol. 217, no. 1, pp. 94–107, 2012.

[20] N. Labadie, R. Mansini, J. Melechovskỳ, and R. W. Calvo, "The team orienteering problem with time windows: An lp-based granular variable neighborhood search," *Eur. J. Oper. Res.*, vol. 220, no. 1, pp. 15–27, 2012.

[21] B. Aghezzaf and H. E. Fahim, "The multi-constraint team orienteering problem with time windows in the context of distribution problems: A variable neighborhood search algorithm," in *Proc. Int. Conf. Logistics Oper. Manage.*, 2014, pp. 155–160.

[22] W. Souffriau, P. Vansteenwegen, G. V. Berghe, and D. Van Oudheusden, "A path relinking approach for the team orienteering problem," *Comput. Oper. Res.*, vol. 37, no. 11, pp. 1853–1859, 2010.

[23] N. Labadie, J. Melechovskỳ, and R. W. Calvo, "Hybridized evolutionary local search algorithm for the team orienteering problem with time windows," *J. Heuristics*, vol. 17, no. 6, pp. 729–753, 2011.

[24] A. Singh, A. Krause, C. Guestrin, and W. J. Kaiser, "Efficient informative sensing using multiple robots," *J. Artif. Intell. Res.*, vol. 34, pp. 707–755, 2009.

[25] J. Binney, A. Krause, and G. S. Sukhatme, "Informative path planning for an autonomous underwater vehicle," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 4791–4796.

[26] L. Heng, A. Gotovos, A. Krause, and M. Pollefeys, "Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 1071–1078.

[27] J. Binney and G. S. Sukhatme, "Branch and bound for informative path planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 2147–2154.

[28] S. Frolov, B. Garau, and J. Bellingham, "Can we do better than the grid survey: Optimal synoptic surveys in presence of variable uncertainty and decorrelation scales," *J. Geophysical Res., Oceans*, vol. 119, no. 8, pp. 5071–5090, 2014.

[29] P. Tokekar, J. Vander Hook, D. Mulla, and V. Isler, "Sensor planning for a symbiotic UAV and UGV system for precision agriculture," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 5321–5326.

[30] N. Azi, M. Gendreau, and J.-Y. Potvin, "An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles," *Eur. J. Oper. Res.*, vol. 202, no. 3, pp. 756–763, 2010.

[31] M. Popović, G. Hitz, J. Nieto, I. Sa, R. Siegwart, and E. Galceran, "Online informative path planning for active classification using uavs," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 5753–5758.

[32] K.-C. Ma, L. Liu, and G. S. Sukhatme, "An information-driven and disturbance-aware planning method for long-term ocean monitoring," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 2102–2108.

[33] R. Pěnička, J. Faigl, P. Váňa, and M. Saska, "Dubins orienteering problem," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 1210–1217, Apr. 2017.

[34] R. Pěnička, J. Faigl, P. Váňa, and M. Saska, "Dubins orienteering problem with neighborhoods," in *Proc. Int. Conf. Unmanned Aircraft Syst.*, 2017, pp. 1555–1562.

[35] S. Arora and S. Scherer, "Randomized algorithm for informative path planning with budget constraints," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 4997–5004.

[36] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[37] A. Singh and A. S. Baghel, "A new grouping genetic algorithm approach to the multiple traveling salesperson problem," *Soft Comput.-A Fusion Found., Methodologies Appl.*, vol. 13, no. 1, pp. 95–101, 2009.

[38] G. A. Croes, "A method for solving traveling-salesman problems," *Oper. Res.*, vol. 6, no. 6, pp. 791–812, 1958.

[39] J. Karbowska-Chilinska, J. Koszelew, K. Ostrowski, and P. Zabielski, "Genetic algorithm solving orienteering problem in large networks," in *Proc. KES*, 2012, pp. 28–38.

[40] H. Li and D. Landa-Silva, "An elitist grasp metaheuristic for the multiobjective quadratic assignment problem," in *Evolutionary Multi-Criterion Optimization*. New York, NY, USA: Springer, 2009, pp. 481–494.

[41] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2015. [Online]. Available: http://www.gurobi.com

[42] A. E. Eiben and S. K. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 19–31, 2011.

[43] N. Veček, M. Mernik, B. Filipič, and M. Črepinšek, "Parameter tuning with chess rating system (CRS-tuning) for meta-heuristic algorithms," *Inf. Sci.*, vol. 372, pp. 446–469, 2016.

[44] M. E. Glickman, "Example of the glicko-2 system," Boston University, Boston, MA, USA, 2012.