2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 24–28, 2017, Vancouver, BC, Canada

Towards an Online Heuristic Method for Energy-Constrained Underwater Sensing Mission Planning

Nikolaos Tsiogkas[1], Valerio De Carolis[1] and David M. Lane[1]

***Abstract*— *Autonomous Underwater Vehicles* (AUVs) have been widely used in scientific or industrial operations over the past years. AUV missions require the vehicle to perform a set of actions autonomously and return to the operators. These missions are currently mostly planned offline by the human operators. Existing solutions in commercial planning software usually only provide an estimate of the mission execution time. The uncertain and dynamic underwater environment can have an effect on the mission performance. More time and energy may be required, disallowing successful mission execution. This work proposes the usage of the *correlated orienteering problem* (COP) that maximises the utility of a sensing mission while respecting energy and time constraints. We propose a heuristic-based on *genetic algorithms* (GA) for the solution of the COP. This heuristic is compared against optimal *mixed-integer quadratic programming* (MIQP) solutions. Results show that the quality of the heuristic solution is in the worst tested case 5.5% less than the 1% optimal solutions. The heuristic proves to be at least 3 times more time efficient than the optimal MIQP solutions in the worst case. The heuristic is finally tested on an embedded platform showing its ability to be used on real robotic platforms.**

I. INTRODUCTION

Recent advances in underwater robotics have yielded reliable platforms for scientific exploration and commercial exploitation of the subaquatic world. Autonomous Underwater Vehicles have made such operations reliable, safe, and cost effective by reducing human involvement; who can't naturally operate in the hostile, underwater environment.

Mission planning in modern commercial vehicles is usually performed prior to the mission by the vehicle operators on the surface. This way of mission planning is static and ignores any disturbances that are caused by the dynamic underwater environment. These disturbances can have an effect on the vehicle's performance, thus reducing the scientific output. For example, in areas affected by tides, high currents are present. These high currents have an immediate effect in the energy and time consumption of the vehicle as it tries to complete a mission [1]. Given these effects, there will be cases that a mission is not possible to be completed within specific time and energy requirements.

This work focuses in such cases, where the energy usage of the vehicle is of importance. It introduces energy awareness in the task scheduling phase trying to improve the performance of the vehicle. The mission of the vehicle consists of sampling a user defined constrained area for scalar field estimation. This is performed by visiting various *inspection points* (IPs) in the aforementioned area to perform sampling. Additionally, the vehicle starts from a user defined starting point and has to reach a specific point to be extracted at the end of the mission. The whole mission has to be performed respecting a maximum energy usage limit. Aim of the mission is to maximise the data gathered while respecting all the aforementioned constraints.

A generic method for maximising the amount of visited IPs, while respecting some budget constraint, is called the *orienteering problem* and is inspired from the sport of orienteering [2]. It was first presented in [3] as the selective travelling salesman problem. It has many applications to transportation and logistics systems [4], [5].

An exact solution method is proposed in [3]. It uses a branch-and-bound technique to find the optimal solution to a given problem instance. Another exact solution is presented in [6]. This work expands the work of [3] by adding valid inequalities to the problem definition. To find the optimal solution it uses a branch-and-cut method.

Given the hard nature of the problem, many heuristic-based solution approaches have been introduced. In [2] a five step heuristic is presented. A Tabu search based heuristic is presented in [7]. In [8] an Ant-Colony optimisation algorithm is presented and compared against a Tabu search heuristic. Another Ant-Colony optimisation method is presented in [9] it is used for the solution of multi-objective orienteering problem instances. In [10] the first *genetic algorithms* (GA) approach is presented. A solution to the generic orienteering problem using GA is presented in [11]. In [12] a GA is used for solving large orienteering problem instances. In [13] a discrete strengthened particle swarm optimisation method is used. A Multi-level variable neighbourhood search is used in [14]. In [15] a greedy randomised adaptive search procedure with path relinking is used. Finally, in [16] a memetic greedy randomised adaptive search procedure is presented.

The orienteering problem has been used in various works regarding sampling or information gathering. In [17] a submodular OP is used to maximise information gathering. It is applied to *Autonomous Surface Vehicle* (ASV) routing for lake and river monitoring. Solution is provided with approximation methods. In [18] a solution for maximising the additional information gain in a sensor network is presented. A submodular OP is used and solution is given using a recursive greedy algorithm. The choice of algorithm was based on the speed and quality of the solutions. In the work presented in [19], the submodular OP is found to be slow for real-time use in large graphs. The paper proposes the use of a linear approximation for area coverage applications. The method was tested in an area coverage scenario performed

[1] Ocean Systems Lab, Heriot Watt University, Edinburgh UK {nt95, vd63, d.m.lane}@hw.ac.uk

by a micro-UAV. In [20] scalar field estimation is performed using a Gaussian process and a branch and bound method to optimise the path. They found that their proposed method has exponential complexity. In [21] a sampling method for marine applications is presented. It tries to maximize the visits to areas of high uncertainty. The solution is acquired using a linearised version of a non-linear function as an orienteering problem instance and a GA based heuristic. In [22] a multi-robot sensing system is demonstrated. The system is designed for agriculture monitoring applications. It utilises an OP instance to maximise visits to potentially misclassified areas. It uses 4-approximation algorithm solution. This solution guarantees that the plan will visit at least a quarter of the points that would produced in an optimal solution.

In [23] a variant of the OP was presented for persistent monitoring tasks using Unmanned Aerial Vehicles. This variant takes into account the correlation in the information sensed from neighbouring inspection points, and is called the *Correlated Orienteering Problem* (COP). In [24] this method was adapted for the underwater survey scenario described before. It was compared with instances of the standard OP and an instance of the *Open Vehicle Routing Problem* (OVRP) [25]. It was found that it performs better than the OPO and the OVRP.

A major drawback of the COP, despite the improvement in the schedule quality, is the complexity of solving the problem exactly. As presented in [24], solving the problem even for small instances can be really time and resource consuming making the online use of it prohibitive. This paper focuses on improving the solution time by using a suboptimal GA based heuristic, while maintaining a high solution quality. This way the vehicle can find a close to optimal plan in a really short time and with minimal computational resources. This plan can be then easily adapted, once more information regarding the environment is gathered, allowing for better mission performance. Time and computational complexity is of essence in an AUV as the processing power of the vehicle can be limited due to heat and energy consumption factors.

The rest of the paper is organised as follows. Section II presents the COP and the GA heuristic solution proposed in this paper. Section III describes the experimental setup and the results obtained. Finally, section IV concludes the paper with remarks to future work.

II. METHODOLOGY

This section presents the methods used to reach the proposed solution. In subsection II-A a method for scalar field estimation using Gaussian processes is presented. In subsection II-B the COP problem for underwater sampling missions is detailed. In II-C the GA heuristic that is proposed is described in detail.

A. Scalar field estimation with Gaussian processes

As mentioned in section I, aim of the vehicle is to estimate a scalar field in a given area. A method for that is presented in [20] and is based on *Gaussian processes* (GPs) for constructing a probabilistic model of the field. The field is approximated by a grid of sampling points. These points are the ones the vehicle has to visit and take measurements.

A GP models the field values as a multivariate Gaussian distribution. The distribution has one dimension for each variable. The covariance of the distribution is calculated using a kernel function. The squared exponential kernel, as shown in (1), is commonly used for that purpose.

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

This kernel shows how fast the correlation of two points reduces with the distance. This is represented by the parameter l and is estimated from collected data. Generally as the parameter grows, the the distance where two points are correlated becomes larger. This correlation is what the COP is taking into account while producing a solution that maximises the information gained.

B. Correlated Orienteering Problem

The correlated orienteering problem, as presented in [23], tries to maximise the following objective function:

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

The objective function is the accumulated rewards of all the visited vertices in a given solution. The set V includes all the vertices. The reward for visiting a single vertex i is described by variable r_i. The variable x_i is a binary variable showing that a vertex i is visited in a solution. As discussed in section I the COP assumes that visiting a vertex senses information regarding vertices around it. N_i represents the neighbourhood of vertex i, and includes all the aforementioned vertices. The reward of collecting this information is added to the reward obtained by visiting vertex i. It is computed as the sum of the rewards for visiting each vertex j in the neighbourhood, multiplied by a weight w_{ij}. This weight depends on vertices i and j. The quadratic term $x_i(x_i - x_j)$ ensures that extra reward, produced by correlation, is only taken into account for vertices that are not visited in the current solution.

Constraints (3) and (4) ensure that the vehicle must start and finish in user defined points.

$$\sum_{i \in V \backslash \{s\}} x_{is} = \sum_{i \in V \backslash \{f\}} x_{fi} = 0 \tag{3}$$

$$\sum_{i \in V \backslash \{s\}} x_{si} = \sum_{i \in V \backslash \{f\}} x_{if} = x_s = x_f = 1 \tag{4}$$

The binary variable x_{si} denotes that any vertex i is visited right after the starting vertex s. In accordance to that variable x_{if} shows that the finishing vertex is after vertex i. In general x_{ij} is showing that a path exists from vertex i to vertex j. For all the other vertices in the solution constraints (5), (6) and (7) are applied.

$$\sum_{i\in V\setminus\{sf\}} x_{ik} = x_i \leq 1 \qquad \forall k \in V \setminus s,f \tag{5}$$

$$\sum_{i\in V\setminus\{sf\}} x_{ki} = x_i \leq 1 \qquad \forall k \in V \setminus s,f \tag{6}$$

$$\sum_{i\in V\setminus f} x_{ik} = \sum_{i\in V\setminus s} x_{ki} \qquad \forall k \in V \setminus s,f \tag{7}$$

Constraints (5) and (6) make sure that a vertex is visited at most once, allowing vertices to be omitted from a solution. To avoid having discontinued paths in a solution, it must be made sure that whenever a vertex is visited, the path must continue to another vertex, with the exception of the user defined finishing vertex. This is ensured by (7). The maximum amount of spent energy is enforced by constraint (8).

$$\sum_{i\in V} x_i c_i + \sum_{j\in V} c_{ij}x_{ij} \leq C_{max} \tag{8}$$

The term $x_i c_i$ is a fixed cost of performing the required sensing operations in vertex i. The cost of travelling from vertex i to the next vertex is added to that, as denoted by $c_{ij}x_{ij}$. It is noteworthy that the fixed cost of operations is not added for the user defined start and finish vertices.

$$u_i - uj + 1 \leq (|V| - 1)(1 - x_{ij}) \quad \forall i,j \in V, i \neq j \tag{9}$$

$$0 \leq u_i \leq |V| \qquad \forall i \in V \tag{10}$$

Finally, constraints (9) and (10) are subtour elimination constraints, allowing only a single tour to be generated.

C. GA heuristic for the COP

Genetic algorithms have been introduced in the 1960s as a method to introduce the natural selection and evolution , as observed in the nature, to computer systems. It involves a population of chromosomes that go through the stages of natural selection, reproduction and mutation [26]. In the case of the work presented in this paper each chromosome is a candidate solution to the COP instance being solved. A description of the steps taken in the optimisation process can be seen in algorithm 1.

The proposed algorithm is the first presented heuristic for the COP. Therefore, most of the parts are newly presented. In particular, unlike other methods it heavily uses 2-opt optimisation [27] in every solution step. This way it tries to increase the amount of inspection points in the solution. Additionally, it presents a novel mutation method. In previous works using GA for solving the standard orienteering problem a mutation was either adding or removing vertices (genes) from a solution. We instead propose that a vertex may be beneficial to be swapped with one of its neighbouring vertices. This technique is used whenever a chromosome is very close to its maximum length and new vertices can not be inserted without violating the cost constraints. Moreover, to have better performance the mutations are performed in parallel, taking advantage modern multicore architectures. Finally, it is proposed to further try to optimise the fittest chromosome after the genetic algorithm finishes.

Algorithm 1 Genetic Algorithm for the Correlated Orienteering Problem

```
Input: $TravelCosts, Rewards, MaxCost, Start, Finish$
Output: $FittestChromosome$
1: INITIALISEPOPULATION
2: while $generation \leq MaxGeneration$ do
3:     SELECTNEWPOPULATION
4:     CROSSOVER
5:     MUTATE
6: Select and Optimise $FittestChromosome$
7: return $FittestChromosome$
```

As it can be seen from algorithm 1 the optimisation process consists of five steps. Initially, a population of chromosomes is generated. Then the chromosomes are going through repetitive operations until a maximum number of generations is reached. Each generation starts by selecting a new population based on natural selection process applied on the previous generation. Then, a number of the chromosomes are allowed to reproduce and generate offsprings. Finally, a number of randomly selected individuals are going through a mutation process. When the maximum number of generations is reached the fittest chromosome is selected. Then some final optimisation is performed on that chromosome and it is returned as the solution to the problem. The fitness of each chromosome is calculated as shown in (11), which was proposed in [12].

$$Fitness = Reward^3/cost \tag{11}$$

Algorithm 2 Initialise Population

```
Input: $PopSize, TravelCosts, MaxCost, Start, Finish$
Output: $Population$
1: $Population \leftarrow [\varnothing]$
2: for $i \leftarrow 1, PopSize$ do
3:     $CR \leftarrow [Start]$
4:     $V \leftarrow [1, \ldots, NumVertices]$
5:     while $!Done$ && $V \neq [\varnothing]$ do
6:         $v \leftarrow$ SELECTRANDOM($V$)
7:         if $cost_{CR+v} \leq MaxCost$ then
8:             $CR \leftarrow CR + v$
9:             $V \leftarrow V \setminus v$
10:        else
11:            $Done \leftarrow True$
12:    $CR \leftarrow CR + Finish$
13:    $CR \leftarrow$ TWOOPT($CR$)
14:    $Population \leftarrow Population + CR$
15: return $Population$
```

Algorithm 2 describes the first step in the GA heuristic method, namely, the population initialisation. The population is initialised as an empty list. It is then populated with a user defined $PopSize$ chromosomes. Each chromosome is generated in the following way. Initially the chromosome contains only the starting vertex. Next, random vertex v is selected from the set V of free vertices. If the current cost of the chromosome plus the cost of travelling to v is lower than the maximum allowed cost $MaxCost$, the vertex is appended to the chromosome. The process is repeated until the $MaxCost$ is reached or there are no more free vertices left. After that the finish vertex is appended to the chromosome. The next step is to perform a *2-opt* optimisation to the chromosome and append it to the population list. This process is repeated $PopSize$ times.

Algorithm 3 Select new population

```
Input: $Population, TourSize$
Output: $NewPopulation$
1: $Pop \leftarrow Population$
2: $NewPopulation \leftarrow [\varnothing]$
3: for $i \leftarrow 1, PopSize$ do
4:     $Candidates \leftarrow$ SELECTRANDOM($Pop, TourSize$)
5:     $Fittest \leftarrow$ GETFITTEST($Candidates$)
6:     $NewPopulation \leftarrow NewPopulation + Fittest$
7: return $NewPopulation$
```

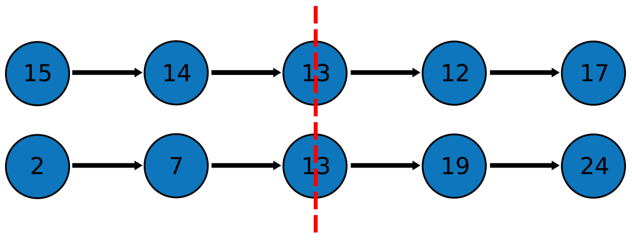
After the population initialisation the main iterative part of the algorithm begins. The first step is the generation of a new population based on a natural selection process. This is performed using a standard technique in the GA literature called *Tournament selection* and presented in algorithm 3. This algorithm starts by an empty new population. It is then populated by $PopSize$ tournaments aiming to select candidates based on their fitness. For each tournament $TourSize$ candidates are randomly selected from the old population. The fittest of the candidates is then wining and is appended to the new population.

The next step, following the population generation, is the crossover operation allowing parts of the population to reproduce and generate offsprings. This process is described in algorithm 4. To perform the crossover operation first two individuals are selected randomly from the population. Then the set of common genes of the two parents is calculated. If the set is empty the combination is incompatible for crossover and the operation is not performed. If the set is not empty then a random gene is selected. The two parent chromosomes are then sliced at that gene and two offsprings are generated by swapping the parent's tail parts. The operation can be seen in figure 1.

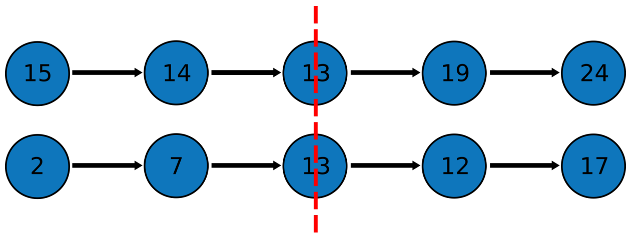
The offsprings are then going through *2-opt* optimisation, any duplicate genes are removed and their feasibility is calculated by comparing them to the $MaxCost$. If both are feasible then their parents are replaced by them in the population. If only one is feasible, it replaces the respective parent and the other parent is replaced by the fittest of the two parents. Finally, if the offsprings are not feasible then the parents are remaining in the population. This process is repeated $NumCrossovers$ times.

Algorithm 4 Population crossover

```
Input: $Population, NumCrossovers$
Output: $Population$
1: $Pop \leftarrow Population$
2: for $i \leftarrow 1, NumCrossovers$ do
3:     $CR^1, CR^2 \leftarrow$ SELECTRANDOM($Pop, 2$)
4:     $I \leftarrow$ GETCOMMONGENES($CR^1, CR^2$)
5:     if $I \neq [\varnothing]$ then
6:         $RG \leftarrow$ SELECTRANDOM($I, 1$)
7:         $OF^1 \leftarrow [CR^1_{[1,RG]}, CR^2_{[RG,end]}]$
8:         $OF^2 \leftarrow [CR^2_{[1,RG]}, CR^1_{[RG,end]}]$
9:         $OF^1 \leftarrow$ TWOOPT($OF^1$)
10:        $OF^2 \leftarrow$ TWOOPT($OF^2$)
11:        $OF^1 \leftarrow$ REMOVEDUPLICATES($OF^1$)
12:        $OF^2 \leftarrow$ REMOVEDUPLICATES($OF^2$)
13:        $Feas^1 \leftarrow$ CHECKFEASIBILITY($OF^1$)
14:        $Feas^2 \leftarrow$ CHECKFEASIBILITY($OF^2$)
15:        if $Feas^1$ && $Feas^2$ then
16:            $Pop_{CR^1} \leftarrow OF^1$
17:            $Pop_{CR^2} \leftarrow OF^2$
18:        else
19:            if $Feas^1$ then
20:                $Pop_{CR^2} \leftarrow$ GETFITTEST($CR^1, CR^2$)
21:                $Pop_{CR^1} \leftarrow OF^1$
22:            else if $Feas^2$ then
23:                $Pop_{CR^1} \leftarrow$ GETFITTEST($CR^1, CR^2$)
24:                $Pop_{CR^2} \leftarrow OF^2$
25:            else
26:                $Pop_{CR^1} \leftarrow CR^1$
27:                $Pop_{CR^1} \leftarrow CR^2$
28: return $Pop$
```



(a) Crossover Parents



(b) Crossover Offsprings

Fig. 1. Two chromosomes before and after the crossover operation. The two parents have a common gene. The operation takes place by swapping the paths after that gene.

Algorithm 5 Population mutation

```
Input: $Population, PopMut, CrMut, AddProb$
Output: $Population$
 1: $Pop \leftarrow Population$
 2: $CR \leftarrow$ SELECTRANDOM($Pop, PopMut$)
 3: for $i \leftarrow 1, PopMut$ do
 4:     $M \leftarrow$ TWOOPT($CR_i$)
 5:     $M \leftarrow$ REMOVEDUPLICATES($M$)
 6:     $V \leftarrow [1, \dots, NumVertices]$
 7:     $FV \leftarrow V \setminus M$
 8:     for $j \leftarrow 1, CrMut$ do
 9:         $p \leftarrow$ GETRANDOM($0, 1$)
10:         if $p \leq AddProb$ then
11:             if $Cost_M \geq 0.99 * max\_cost$ then
12:                 $v \leftarrow$ SELECTRANDOM($M, 1$)
13:                 $n \leftarrow$ GETMAXFREENEIGHBOUR($M_v$)
14:                 if $Fit_{M_v} \leq Fit_{M_n}$ then
15:                     $M_v \leftarrow M_n$
16:             else
17:                 if $FV \neq [\varnothing]$ then
18:                     $v \leftarrow$ SELECTRANDOM($FV, 1$)
19:                     $idx \leftarrow$ FINDBESTINSERTION($v, M$)
20:                     $M \leftarrow M_{[\dots, idx, v, idx+1, \dots]}$
21:         else
22:             if $cost_M \geq 0.99 * max\_cost$ then
23:                 $v \leftarrow$ GETMINLOSSGENE(M)
24:                 $M \leftarrow M_{[\dots, v-1, v+1, \dots]}$
25:     $Pop_{CR_i} \leftarrow M$
26: return $Pop$
```

The final step in the iterative part of the GA optimisation process is the mutation step. It is described in algorithm 5. Initially $PopMut$ chromosomes are randomly selected from the population. Then each chromosome goes through the following mutation process. The first step applies a *2-opt* and removes any duplicate genes. Then, the set of free genes is calculated. In each mutation process $CrMut$ operations take place.

The mutation operations can be either the addition or the removal of a gene with some probability. The addition operation has two behaviours depending on the cost of the chromosome. The first behaviour occurs when the chromosome has a cost close to the $MaxCost$. In that case a random gene is selected from the chromosome. A search over the free neighbours of that chromosome is performed and the fitness of replacing the selected gene with its neighbour is calculated. If that fitness is higher than the current fitness the mutation happens. The second behaviour, happening in chromosomes with low cost, selects a random gene from the free genes. Then it finds the best position for insertion and inserts it there. The removal operation happens only in chromosomes with cost close to the $MaxCost$. In those cases the gene whose removal causes the minimum fitness loss is selected. It is then removed from the chromosome.

TABLE I
GENETIC ALGORITHM PARAMETERS

Parameter	Values
$MaxGeneration$	50
$PopSize$	200
$TourSize$	3
$NumCrossovers$	20
$PopMut$	50
$CrMut$	10
$AddProb$	0.9

After the maximum number of generations is reached the fittest chromosome is selected. On that chromosome a final optimisation round is applied. In this last round each gene is traversed and is checked if replacing it with any of its neighbours results in higher fitness feasible chromosome. If yes a swap is performed. After this final step the chromosome is the solution to the COP instance.

III. EXPERIMENTAL RESULTS

The performance of the GA proposed heuristic is compared against the *Mixed Integer Quadratic Programming* (MIQP) formulation of the COP. The heuristic is evaluated by comparing the expected accumulated utility and the planning time against the exact solution.

The planning is performed on a simulated sampling mission an AUV, like Nessie VII AUV [28], has to perform. The simulated mission consists of multiple *sampling points*, arranged in a grid. Additionally, the vehicle is provided with specific starting and finishing points. Moreover, the vehicle is given a limited amount of energy to complete its tasks. The overall vehicle's goal is to maximise the gained utility before reaching the energy threshold.

Simulations demonstrate the effect of the above methods on the planning process. To solve the MIQP formulation the Gurobi 6.5 commercial solver [29] is used. The GA heuristic method is implemented in C++11 and compiled using g++ version 5.3.0 on Ubuntu 14.04. The GA parameters are presented in table I. The experiments were run on a 2.6GHz Intel Core i5-3320M processor having 16GB of RAM. It should be noted that the actual hardware setup of Nessie VII AUV uses an embedded PC104 solution with much less computational power. Source code for both approaches can be found online [1] [2].

The performance evaluation of the proposed heuristic method required several tests to be run. The cost of movement was set to be directly proportional to the distance that had to be travelled. Moreover, the vehicle incurs a fixed sensing cost of one unit for each inspection point that was visited. As mentioned in II-A the correlation between two points is represented by the kernel function described in (1). For the purpose of this paper the value of l was set to 2. This kernel function is used to calculate a weight for equation (2). An example of the maximum correlation range can be seen in figure 2.

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

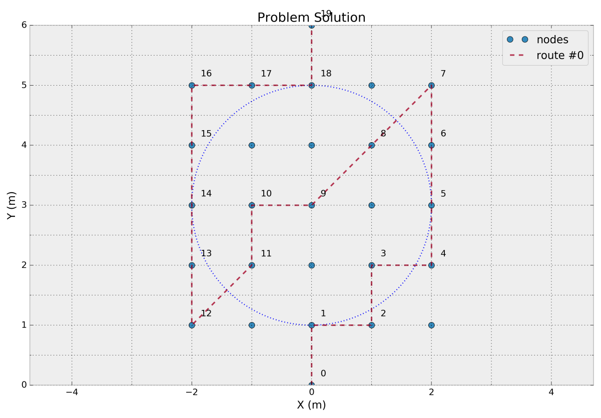


Fig. 2. Correlation range from inspection point 9. All the non-visited points inside the blue circle contribute to correlation utility to point 9. This represents the information we gain about the other points by sampling that inspection point.

The tests that were run involved two different runs of the MIQP solver. The first was performed with 1% optimality gap, meaning that the output value is 1% different from the optimal. The second was performed using a 5% gap. These were used to limit the execution time of the MIQP to reasonable times. Additionally there was a maximum allowed amount of time that the solver was allowed to run, and it was set to 36000 seconds. Regarding the GA, 1000 trials were executed for each size and energy budget. Then statistical results were extracted.

As mentioned in the previous section a point of evaluation is the expected accumulated utility of each method. To evaluate that five different sizes of grid and four different energy budgets were chosen. The energy budgets were computed by finding the optimal path to traverse all the vertices using an OVRP instance [25]. To that all the required sensing costs for each vertex were added. The required budgets are then calculated by reducing the maximum budged accordingly. The results of the various tests can be seen in table II.

As expected the 1% optimal MIQP instance performs usually better. The heuristic performance can be mostly compared to the 5% MIQP solution. They perform on average really closely. By examining the standard deviation of the heuristic it can be seen that it can find solutions close to the 1% optimal given enough tries. In figure 3 the 1% optimal calculated path for a 5x5 grid with 75% available energy can be seen. The blue color represents the area coverage due to correlation. The color is darker closer to the actually sensed points and fades out as the sensor reading quality drops. The heuristic solution for the same method can be seen in figure 4. They achieve the same utility. In terms of coverage quality the MIQP achieves to cover the whole area, while the GA performed better coverage on the central part of the area. It should be noted that the GA gives a solution similar to a standard lawnmower pattern used by vehicles in the field.

The second criterion of evaluation is the path planning time. Given that it is required for the vehicle to perform its mission autonomously it is essential that the mission can be planned online and as fast as possible. The results can be seen in table III.

As it can be seen the proposed heuristic performs much better than the 1% MIQP solutions by several orders of magnitude. It also outperforms the 5% gap solution. That is even more clearly visible in larger instances where again it is many times faster. The execution time is ideal for online usage. Since the heuristic is time-efficient it can be run multiple times in order to find the best solution. This solution, in many cases, will be really close to the optimal, but in planning time it will be orders of magnitude faster. The benefits of the heuristic will be even more visible in

TABLE II
UTILITY GAINED FOR DIFFERENT GRID SIZES AND BUDGETS

Grid Size	Algorithm	Budget			
		100%	75%	50%	25%
5x5	COP 1%	24.554	**20.771**	**14.725**	**7.108**
	COP 5%	24.377	20.216	14.571	**7.108**
	GA	**25**	20.704	14.486	7.070
	GA St.Dev.	0	0.087	0.287	0.090
6x6	COP 1%	**35.554**	**30.056**	**21.633**	**10.771**
	COP 5%	35.377	30.017	21.364	**10.771**
	GA	**35.554**	29.758	21.210	10.526
	GA St.Dev.	0	0.144	0.281	0.323
7x7	COP 1%	**48.554**	**41.81**	**30.304**	**15.079**
	COP 5%	47.285	41.418	29.849	**15.079**
	GA	**48.554**	40.716	29.013	14.606
	GA St.Dev.	0	0.389	0.422	0.311
8x8	COP 1%	63.377	**54.426**	**39.828**	**20.095**
	COP 5%	62.754	54.165	39.343	**20.095**
	GA	**63.547**	53.194	37.969	19.110
	GA St.Dev.	0.053	0.413	0.549	0.64
9x9	COP 1%	80.377	**69.666**	**51.207**	**25.835**
	COP 5%	79.662	67.804	50.244	**25.835**
	GA	**80.456**	67.624	48.395	24.723
	GA St.Dev.	0.226	0.474	0.791	0.719

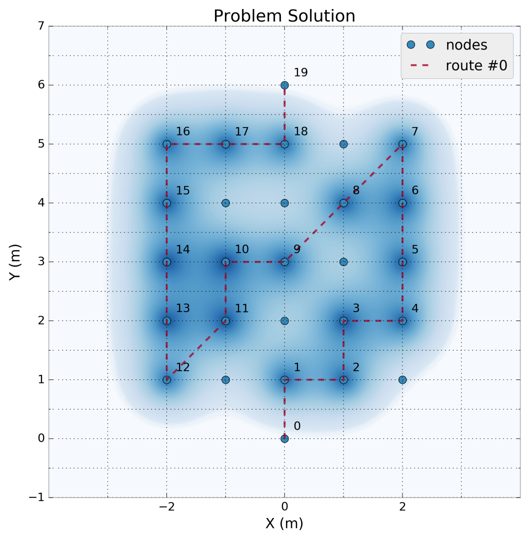


Fig. 3. MIQP problem solution for a 5x5 grid and energy budget of 38.25 units (75% of maximum). The utility gathered was 20.701.

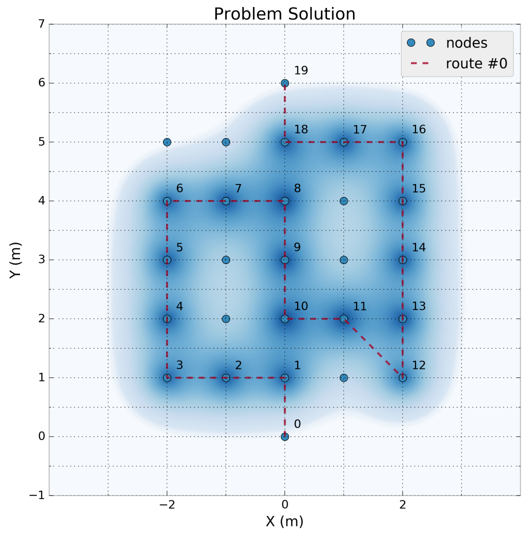


Fig. 4. GA problem solution for a 5x5 grid and energy budget of 38.25 units (75% of maximum). The utility gathered was 20.704.

TABLE III
COMPUTATION TIME(S) FOR DIFFERENT GRID SIZES AND BUDGETS

Grid Size	Algorithm	Budget 100%	75%	50%	25%
5x5	COP 1%	1.381	5.118	14.981	1.257
	COP 5%	0.514	3.664	7.979	1.076
	GA	**0.221**	**0.166**	**0.127**	**0.101**
	GA St.Dev.	0.03	0.013	0.013	0.014
6x6	COP 1%	34.745	13.118	389.658	22.541
	COP 5%	0.898	1.559	70.983	9.001
	GA	**0.35**	**0.243**	**0.159**	**0.09**
	GA St.Dev.	0.06	0.028	0.017	0.005
7x7	COP 1%	2.42	190.585	2492	560.24
	COP 5%	1.926	2.505	97.181	27.68
	GA	**0.785**	**0.411**	**0.207**	**0.103**
	GA St.Dev.	0.123	0.06	0.022	0.004
8x8	COP 1%	25.15	53.98	28965	5715.92
	COP 5%	11.78	13.91	272.9	97.93
	GA	**1.042**	**0.636**	**0.298**	**0.129**
	GA St.Dev.	0.197	0.077	0.023	0.015
9x9	COP 1%	214.77	518.53	MAX	MAX
	COP 5%	92.247	23.041	865.83	408.81
	GA	**2.038**	**1.1**	**0.452**	**0.165**
	GA St.Dev.	0.361	0.147	0.052	0.017

lower-end or embedded processors used in many autonomous systems.

To evaluate the performance on a lower end system, the heuristic algorithm was tested on the ARM processor of an Nvidia JETSON TK1 [30]. This choice was made because such processors can be easily embedded in any kind of robotics project due to their low power consumption and good performance. As an example one can refer to the commercial drone manufacturer DJI and their Manifold onboard computing solution [31]. A TK1 system, built by Nvidia, is currently integrated into the Nessie VII vehicle.

The online applicability of the proposed heuristic method

TABLE IV
UTILITY GAINED FOR DIFFERENT GRID SIZES AND BUDGETS USING AN EMBEDDED ARM PROCESSOR

Grid Size	Processor	Budget 100%	75%	50%	25%
5x5	GA x86	**25**	20.704	14.486	7.070
	GA x86 St.Dev.	0	0.087	0.287	0.090
	GA ARM	24.999	**20.709**	**14.512**	**7.073**
	GA ARM St.Dev.	0.014	0.085	0.278	0.086
6x6	GA x86	**35.554**	**29.758**	21.210	**10.526**
	GA x86 St.Dev.	0	0.144	0.281	0.323
	GA ARM	35.552	**29.758**	**21.221**	10.513
	GA ARM St.Dev.	0.244	0.151	0.280	0.326
7x7	GA x86	**48.554**	40.716	**29.013**	14.606
	GA x86 St.Dev.	0	0.389	0.422	0.311
	GA ARM	**48.554**	**40.718**	28.989	**14.626**
	GA ARM St.Dev.	0	0.387	0.589	0.305
8x8	GA x86	63.547	**53.194**	37.969	19.110
	GA x86 St.Dev.	0.053	0.413	0.549	0.640
	GA ARM	**63.550**	53.151	**37.981**	**19.141**
	GA ARM St.Dev.	0.039	0.422	0.813	0.643
9x9	GA x86	80.456	67.624	48.395	24.723
	GA x86 St.Dev.	0.226	0.474	0.791	0.719
	GA ARM	**80.471**	**67.628**	**48.421**	**24.767**
	GA ARM St.Dev.	0.21	0.492	0.791	0.697

was investigated by executing the same set of 1000 runs on the embedded computer. In these runs the results obtained with the normal x86 processor are compared to those of the ARM board. The results can be seen in the following tables. In table IV one can see the average utility obtained by the solution on each processor type, while in table V the average time for each solution is presented.

As it can be seen the solution quality is not affected by the processor architecture and performance. The difference utility obtained is in the margins of statistical error, while in a few cases it is exactly the same. This result is expected as nothing algorithmically changed.

Table V presents more interesting results as the time efficiency in an embedded system is crucial for the online applicability of the proposed heuristic. As it can be seen the heuristic performs well on the embedded processor. On average the execution time is two to three times higher but still in many cases it is lower than the time the 5% MIQP achieved on the x86 processor. These results demonstrate the applicability of the heuristic method to systems that can be easily embedded into any robotic platform.

IV. CONCLUSION

The problem of online task scheduling for sensing applications using AUVs is studied in this work. It proposes a genetic algorithm-based heuristic for the solution of the correlated orienteering problem. The COP is ideal for sensing applications as it considers the correlations of the sensed information in various sensing points while keeping the time or energy usage constrained. The heuristic method is compared to a mixed integer quadratic programming solution. In terms of information quality the heuristic performs close to the near-optimal MIQP solutions. The real benefit of the heuristic

TABLE V

COMPUTATION TIME(S) FOR DIFFERENT GRID SIZES AND BUDGETS USING AN EMBEDDED ARM PROCESSOR

Grid Size	Processor	Budget			
		100%	75%	50%	25%
5x5	GA x86	**0.221**	**0.166**	**0.127**	**0.101**
	GA x86 St.Dev.	0.030	0.013	0.013	0.014
	GA ARM	0.627	0.471	0.362	0.291
	GA ARM St.Dev.	0.034	0.021	0.018	0.017
6x6	GA x86	**0.350**	**0.243**	**0.159**	**0.090**
	GA x86 St.Dev.	0.06	0.028	0.017	0.005
	GA ARM	0.972	0.674	0.449	0.285
	GA ARM St.Dev.	0.161	0.066	0.026	0.007
7x7	GA x86	**0.785**	**0.411**	**0.207**	**0.103**
	GA x86 St.Dev.	0.123	0.060	0.022	0.004
	GA ARM	2.061	1.115	0.589	0.322
	GA ARM St.Dev.	0.335	0.157	0.048	0.007
8x8	GA x86	**1.042**	**0.636**	**0.298**	**0.129**
	GA x86 St.Dev.	0.197	0.077	0.023	0.015
	GA ARM	2.752	1.664	0.813	0.384
	GA ARM St.Dev.	0.510	0.191	0.046	0.024
9x9	GA x86	**2.038**	**1.100**	**0.452**	**0.165**
	GA x86 St.Dev.	0.361	0.147	0.052	0.017
	GA ARM	5.412	2.748	1.220	0.467
	GA ARM St.Dev.	0.962	0.361	0.138	0.027

is its time complexity which is orders of magnitude lower than the MIQP. This makes it ideal for online scheduling of tasks. The most imminent future prospect is the extension of the heuristic to multi-robot teams. Additionally, since the algorithm is generic enough it would be interesting to see it integrated in other robotic platforms than AUVs. An example would be aerial drones taking air quality measurements while disturbed by windfields. Finally, given the capabilities provided by the embedded GPU of the Nvidia Jetson TK1, it would be interesting to integrate the proposed heuristic with it, as research has shown that a GPU can vastly improve the performance of genetic algorithms [32].

REFERENCES

[1] J. Bellingham and J. Willcox, "Optimizing auv oceanographic surveys," in *Autonomous Underwater Vehicle Technology, 1996. AUV '96., Proceedings of the 1996 Symposium on*, Jun 1996, pp. 391–398.

[2] I.-M. Chao, B. L. Golden, and E. A. Wasil, "A fast and effective heuristic for the orienteering problem," *European Journal of Operational Research*, vol. 88, no. 3, pp. 475–489, 1996.

[3] G. Laporte and S. Martello, "The selective travelling salesman problem," *Discrete applied mathematics*, vol. 26, no. 2-3, pp. 193–207, 1990.

[4] P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden, "The orienteering problem: A survey," *European Journal of Operational Research*, vol. 209, no. 1, pp. 1–10, 2011.

[5] A. Gunawan, H. C. Lau, and P. Vansteenwegen, "Orienteering problem: A survey of recent variants, solution approaches and applications," *European Journal of Operational Research*, 2016.

[6] M. Gendreau, G. Laporte, and F. Semet, "A branch-and-cut algorithm for the undirected selective traveling salesman problem," *Networks*, vol. 32, no. 4, pp. 263–273, 1998.

[7] M. Gendreau, G. Laporte, and F. Semet, "A tabu search heuristic for the undirected selective travelling salesman problem," *European Journal of Operational Research*, vol. 106, no. 2, pp. 539–545, 1998.

[8] Y.-C. Liang, S. Kulturel-Konak, and A. E. Smith, "Meta heuristics for the orienteering problem," in *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, vol. 1. IEEE, 2002, pp. 384–389.

[9] M. Schilde, K. F. Doerner, R. F. Hartl, and G. Kiechle, "Metaheuristics for the bi-objective orienteering problem," *Swarm Intelligence*, vol. 3, no. 3, pp. 179–201, 2009.

[10] M. F. Tasgetiren, "A genetic algorithm with an adaptive penalty function for the orienteering problem," *Journal of Economic and Social Research*, vol. 4, no. 2, pp. 1–26, 2001.

[11] X. Wang, B. L. Golden, and E. A. Wasil, "Using a genetic algorithm to solve the generalized orienteering problem," in *The vehicle routing problem: latest advances and new challenges*. Springer, 2008, pp. 263–274.

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

[13] Z. Sevkli and F. E. Sevilgen, "Discrete particle swarm optimization for the orienteering problem," in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[14] Y.-C. Liang, S. Kulturel-Konak, and M.-H. Lo, "A multiple-level variable neighborhood search approach to the orienteering problem," *Journal of Industrial and Production Engineering*, vol. 30, no. 4, pp. 238–247, 2013.

[15] V. Campos, R. Martí, J. Sánchez-Oro, and A. Duarte, "Grasp with path relinking for the orienteering problem," *Journal of the Operational Research Society*, vol. 65, no. 12, pp. 1800–1813, 2014.

[16] Y. Marinakis, M. Politis, M. Marinaki, and N. Matsatsinis, "A memetic-grasp algorithm for the solution of the orienteering problem," in *Modelling, Computation and Optimization in Information Systems and Management Sciences*. Springer, 2015, pp. 105–116.

[17] A. Singh, A. Krause, C. Guestrin, and W. J. Kaiser, "Efficient informative sensing using multiple robots," *Journal of Artificial Intelligence Research*, pp. 707–755, 2009.

[18] J. Binney, A. Krause, and G. S. Sukhatme, "Informative path planning for an autonomous underwater vehicle," in *Robotics and automation (icra), 2010 IEEE international conference on*. IEEE, 2010, pp. 4791–4796.

[19] L. Heng, A. Gotovos, A. Krause, and M. Pollefeys, "Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1071–1078.

[20] J. Binney and G. S. Sukhatme, "Branch and bound for informative path planning," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 2147–2154.

[21] 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," *Journal of Geophysical Research: Oceans*, vol. 119, no. 8, pp. 5071–5090, 2014.

[22] P. Tokekar, J. Vander Hook, D. Mulla, and V. Isler, "Sensor planning for a symbiotic uav and ugv system for precision agriculture," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 5321–5326.

[23] J. Yu, M. Schwager, and D. Rus, "Correlated orienteering problem and its application to informative path planning for persistent monitoring tasks," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, Sept 2014, pp. 342–349.

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

[25] F. Li, B. Golden, and E. Wasil, "The open vehicle routing problem: Algorithms, large-scale test problems, and computational results," *Computers & Operations Research*, vol. 34, no. 10, pp. 2918–2930, 2007.

[26] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

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

[28] N. Valeyrie, F. Maurelli, P Patron, J. Cartwright, B. Davis, Y. Petillot, "Nessie v turbo: a new hover and power slide capable torpedo shaped auv for survey, inspection and intervention," in *AUVSI North America 2010 Conference*, 2010.

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

[30] N. Corporation, "Nvidia jetson tk1," 2017. [Online]. Available: https://goo.gl/ObH9PC

[31] DJI, "Dji manifold," 2015. [Online]. Available: https://goo.gl/Diwg7u

[32] P. Pospichal, J. Jaros, and J. Schwarz, "Parallel genetic algorithm on the cuda architecture," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2010, pp. 442–451.