A genetic planner for mission planning of cooperative agents in an underwater environment

Branko Miloradović, Baran Çürüklü, and Mikael Ekström
School of Innovation, Design, and Engineering
Mälardalen University
Västerås, Sweden

***Abstract*—In this paper a Genetic Algorithm (GA) is used for solving underwater mission planning problem. The proposed genetic planner is capable of utilizing multiple Autonomous Underwater Vehicles (AUVs) and Remotely Operated Vehicles (ROVs) in a mission plan, as well as running multiple tasks in parallel on the agent's level. The problem is described using STRIPS modeling language. The proposed planner shows high robustness regarding initial population set, which is randomly generated. Chromosomes have variable length, consisting of active and inactive genes. Various genetic operators are used in order to improve convergence of the algorithm. Although genetic planner presented in this work is for underwater missions, this planning approach is universal, and it is not domain dependent. Results for a realistic case study with five AUVs and almost 30 tasks show that this approach can be used successfully for solving complex mission planning problems.**

Keywords— High-Level Planning, Genetic Algorithms, Mission Planning, Multi-Agent Systems, Underwater robotics

I. INTRODUCTION

A. Formal description languages

In artificial intelligence (AI) given a formal description of the behaviour of a set of actions, to find a sequence of actions which lead from a known state of the world to a desired one is defined as planning [1]. In accordance with this definition, a simplified model of the world is needed. This simplified model is generally represented as a set of predicates, or simply rules, expressing possible states, and a set of actions that can act on these predicates. Perhaps, the most popular formal language used for expressing automated planning problems is STRIPS [2]. STRIPS is a quadruple (C, O, S, G) in which elements have the following meanings:

- C is the set of conditions.
- O is the set of actions, where each action has its preconditions (what must be done before that action) and post conditions (effects of that action on the world state).
- S is the initial state, with a set of conditions that are true, and all others are assumed to be false.
- G is the goal state, with set of conditions that are either true or false.

Inspired by STRIPS and ADL (Action Description Language) Planning Domain Definition Language (PDDL) [3] was developed in an attempt to standardize AI planning languages. PDDL separates the planning problem into two main parts: domain description and problem description. Domain description consists of requirements, predicates, and actions, where actions have parameters, preconditions, and effects. Problem description consists of objects, initial conditions, and goal-states.

B. Planning domain

Here, two main levels of abstraction are considered. Firstly, the High-Level Planning (HLP) that allows the user to describe different tasks regarding operations performed without specifying the exact actions agent needs to perform. The output of the HLP is usually a mission plan, consisting of the tasks that certain agent needs to perform. Secondly, Low-Level Planning (LLP) that is typically carried out at the agent level and includes generation of waypoints, actions and other similar low level (LL) tasks implemented by the programmer. E.g. *follow the human* is a high-level (HL) task while generating a set of waypoints for the movement of the agent following the human is considered an LL task. A set of LL tasks, or as they are often called in the literature – actions, form a HL task. Since in this paper, a problem of mission planning is tackled (high-level of abstraction), in the further text HL tasks are referred to as just "task".

C. Problem formulation

In the proposed solution given a global mission objective, constraints, and available resources (agents), the planning activity has to divide the global mission objectives into several sub-goals. The planner then needs to assign an agent taken individually, or a set of agents behaving as a team or a swarm to each sub-goal and schedule these more basic activities on time. These agents does not necessarily have to be homogenous as task specialized agents can be in this set too. The output of the planner should be a global plan of concurrent activities (schedule) for the set of selected agents with temporal constraints. This type of problem is a mixed planning/scheduling problem which well belongs to the more general domain of combinatorial problems.

In this paper for tackling this problem an evolutionary algorithm, more specifically a modification of Holland's simple genetic algorithm [4], is proposed. The binary data encoding is replaced with integers, thus the appropriate changes are made in order to utilize existing genetic operators.

The taxonomy is a very useful as the first separation of different classes of problem setting. In [5] a formal analysis of multi-robot task allocation (MRTA) is given, and three different axes for use in describing MRTA problems are proposed. The genetic planner presented in this work covers all of these problems, except for multi-robot (MR) tasks. MR tasks are tasks that can require more than one robot for its fulfilment. Furthermore, the problem described here belongs to the class of NP-hard optimization problems. It is important to note that some simplifications are assumed, like atomic time, deterministic effects, and omniscience – world state is completely known.

Since much of this research is done in correlation with the Smart and Networking Underwater Robots in Cooperation Meshes (SWARMs) project, the focus is on the underwater mission planning, using different (AUVs) and (ROVs) as agents. Even though the main focus is on underwater missions, the planning approach shown here is generic, and it is not domain dependent.

The presented planner has task parallelism on agent's level and repetition of tasks with different parameters which is not supported by any of the mentioned approaches in the next section. In addition, the initial population is completely random, the planner does not start with a population of only feasible solutions. It is also important to emphasize that multi-agent planning is possible with the proposed genetic planner. Agents and tasks are encoded in a chromosome, and a chromosome is represented as a vector.

The paper is organized as follows: first, an overview of related work is made; then GA is described, following the algorithm implementation; next two sections are experimental results and a case study; finally, conclusion and future work are presented.

II. Related Work

There has been a significant number of different attempts to solve mission planning problems by applying computational approaches. A review of these methods from the military domain can be found in [6]. A distributed hierarchical Petri net model of mission control procedure for describing logical relations among mission level, task level, and behavior level is presented in [7]. The mission level takes on the genetic algorithm to plan and re-plan the global path according to velocity, energy and other mission constraints in unstructured, partially unknown or hostile environments. A mission planning problem formulated as a Constrained Markov Decision Process (CMDP) for finding the optimal policy has been presented in [8]. In addition, there are attempts to solve this problem with heuristic approaches such as tabu search [9] or an evolutionary algorithm. In [10] authors use PDDL modeling language and simple genetic algorithm with several enhancements. Their approach uses variable chromosome length, and it results in efficient memory usage. On the other hand, this approach does not allow usage us multi-agents nor parallelism of tasks.

The problem of multi-objective optimization with the evolutionary algorithm is tackled before, where a way of benchmarking obtained solutions is also addressed in [11]. They have developed an evolutionary planner called Divide-and-Evolve that embeds a classical planner and feeds it with a sequence of subproblems of the problem at hand. A search for a non-dominated solution with objectives being fuel consumption and makespan of a mission is shown in [12] and [13], where the mission planning problem was formulated as a Constraint Satisfaction Problem (CSP). The CSP model defines several constraints, including order and temporal constraints, but this allows only sequential task execution (no parallel tasks on a single agent), and tasks cannot be repeated multiple times. The chromosome encoding is done with two rows vector, where columns represent tasks. The cooperative task planning problem for multiple AUVs carrying out a series of tasks against multiple targets is addressed in [14] . GA was used for finding the optimal mission plan. The mission consists of flying to the designated location and passing over a set of predefined targets. Targets are assumed to be homogenous. A chromosome is encoded as $n \; x \; m$ matrix, where n is the number of agents and m is the number of tasks. The problem here is that n is fixed. This means that the chromosome will always have dimension n, even if the plan uses a number of agents that is less then n. This planner too does not support parallelism of tasks on agent level. Another popular way for solving resource-constrained project scheduling problem (RCPSP) is by using GAs, where chromosome representation is based on random keys as presented in [15]. The schedule is constructed by using a heuristic priority rule in which the priorities and delay times of the activities are defined by the genetic algorithm. This approach does not allow use multi-agents nor parallel tasks on a single agent. A Decomposition Based Genetic Algorithm (DBGA) for RCPSP is proposed in [16]. This method divides the RCPSP problem into smaller problems and obtains the solution for the problem by combining the solution of each such subproblem. It is shown that the decomposition based approach finds satisfactory near-optimal solutions.

III. Genetic Algorithm

A large number of conventional planning algorithms handle planning by mapping the search space into a graph or a tree, searching through the nodes using heuristic functions, cutting infeasible branches or backtracking from dead-ends. GA solves planning problem with a different method, sometimes leading to sub-optimal solutions.

Genetic Algorithms are adaptive heuristic search methods for solving both constrained and unconstrained optimization problems that use concepts from evolutionary biology to search for an optimal solution. GAs belongs to the larger class of evolutionary algorithms (EAs) and are considered as one of the most suitable algorithms to solve a problem for which little is known. GA works by starting with an initial generation of candidate solutions that are tested against the objective function. A number of individuals, defined by the population size, are seeded in the search space. The algorithm tries to calculate the quality of the current candidate solutions, i.e. its distance to the solution. In the next iteration, the algorithm tries to reseed population in the neighbourhood of individuals that are closest to the solution. Subsequent generations evolve from the previous generation through genetic operators. Reseeding of the population across the search space is done according to probabilistic selection, crossover, and mutation. Probabilistic distribution guides the chromosomes through the search space,

unaware of its configuration. This means that search is uninformed, which makes the whole process quite simple, allowing the population set to be large giving better chances of finding a good solution. The goal of retaining the best performing chromosomes from one generation to the next, along with choosing the parents is carried out by selection operator. In this way, the best solutions are favoured in the reproduction process.

After completing the configuration file with all the information needed regarding tasks, a random population of individuals is created. Again, each individual holds a plausible solution to the high-level planning problem. When creating the initial population, feasibility of candidate solutions is not taken into account. This means that the initial population is created completely random consisting of task/agents – active genes, and "dummy" inactive genes. An example of how population set looks like is given in the Fig. 1. Where $A_1, A_2, \ldots A_n$ (red squares) are agent identifiers and $T_1, T_2, \ldots T_n$ (blue squares) are task identifiers in the chromosome. Orange boxes represent inactive "dummy" genes.

Since the length of the mission plan is not known *a priori*, chromosomes have a variable length. A variable chromosome length is used in [10], to the opposite of this approach, a fixed chromosome length with "dummies" filling up space up to the predefined chromosome length is proposed in [19]. In this work a mix of the approaches mentioned above is used, i.e. all chromosomes have the same length, but that length changes based on the longest chromosome. Shorter chromosomes are filled in with inactive genes. This characteristic is bio-inspired since humans as well as other species also have a certain percentage of inactive, so-called "junk" genes. Note that in this work the exact percentage of these genes are not considered, thus an ad hoc solution is proposed (see below).

The number of inactive genes can be set before running the algorithm, i.e. the number of inactive genes added to the longest solution can be set. The other solutions are automatically filled up with inactive genes forming population set as a matrix of $n \; x \; m$, where n is size of a population and m is the number of genes in the chromosome. This means that m is a sum of active and inactive genes. Minimum number of inactive genes is zero, this option is the fastest, but will likely produce worse solution than setting the number of inactive genes to a value higher than zero. This is especially important in the initial population set, where larger number of genes will help algorithm converge in a lesser number of iterations. Genetic operators, crossover and a specific type of a mutation (the so-called shrink mutation), will take care of unnecessary tasks/agents in the candidate solutions and eventually replace them with inactive genes. This process

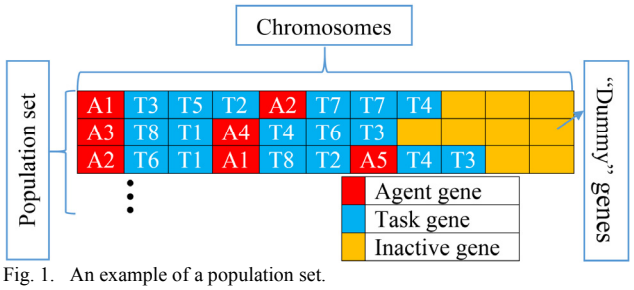

Fig. 1. An example of a population set.

is done through the crossover operator and allows the creation of new offspring by the mating process of previously chosen parents. This operator allows big jumps in the search space by combining several candidate solutions. If the job of the crossover operator is to find better solutions by exploiting current ones, then mutation can be seen as the operator that helps to explore the entire search space. Mutation allows genetic algorithms to avoid falling into local minima by maintaining genetic diversity in the population.

Although there are many different ways of how these operators work, EA/GA are generally very robust algorithms, working very well even with the simplest types of operators. The thing that is more important for obtaining good solutions is the knowledge representation and the way it is encoded into chromosomes. There are many different ways of representing chromosomes as it is shown in [17] and [18]. Bit strings are the most common way, but it is also possible to use integers/floats, lists, hashes, objects or any other existing data structure. In the approach shown in this paper, a gene encodes both agent or task identifier while a chromosome encodes a complete mission plan. Therefore, a search for feasible/best solution becomes a search for the best individual consisting of valid task and agent genes in the population set. The behaviour of the proposed genetic mission planner corresponds to the standard EA paradigm:

- After creating an internal knowledge representation model, an initial random population is created. Each individual represents a possible solution.
- The evolution process consists in selecting individuals, applying genetic operators to them and adding the resulting offspring to the population of the next generation
- A cost value of each individual is then evaluated and updated in the fitness function. The evolution process is repeated until the stopping criterion is met.

IV. Implementation

Inputs to the genetic planner are task constraints, task parameters, and *"main tasks"* (MTs), which is a list containing tasks that need to be executed (e.g. scan seabed, take images, etc.). Task constraints include information about relations between tasks and which tasks are allowed to run in parallel. An example of parallel tasks is to scan seabed and take images at the same time, but it is not possible to scan seabed and use the acoustic modem for communication. Task parameters are characterized by several aspects such as task location, duration, and energy required for task execution consisting of task parameters and pre/post conditions, while agents only have parameters. The structure of a task is shown in the Fig. 2. The cost of the initial population is evaluated in the fitness function.

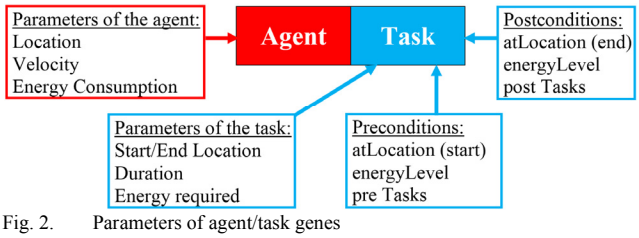

Fig. 2. Parameters of agent/task genes

Depending on if the function is being minimized or maximized, the vector containing population cost is sorted in ascending or descending order. An alternative way used for maximizing optimization is to multiply the cost by -1 and do the minimization. The fitness function will be described in detail in the next section. The iteration process can have different stopping criteria. It can be either cost, a maximum number of iterations reached or it can stop if the global best solution does not improve over a fixed number of iterations. Since the desired cost value is not always known, in this case stopping criteria is the number of iterations.

In order to decide which individuals will be chosen for breeding, selection operator is introduced. In the literature, many different selection methods can be found [18], but the most popular are rank, roulette wheel, and tournament selection. Although all of these selection methods were implemented, rank selection showed the best performance. The rank selection might have slower convergence rate because there is not much of a difference, with respect to selection probability, between the best individuals and the rest of the population. This can also be an advantage of this method because there is a lower chance of a premature convergence i.e. less chance to get stuck in local minima. The population is firstly ranked based on the fitness value of each chromosome. After that, a ranking value is assigned to each individual, so that the best one will have a value of n (number of individuals in the population set), the second one will have $n-1$, etc. and the worst one will have a value of 1. Since all the chromosomes have a chance to be selected, this method ensures good diversity in the population set. In this work using the rank selection operator two equal groups of parents are selected for later mating.

Another selection operator is implemented, and it is called elitist selection. The elitism allows best chromosomes from current generation to be transferred to the next one unaltered. Elitism ensures that algorithm does not diverge, and it guarantees that the quality of the obtained solution will not deteriorate in the next generation. The downside of this selection operator is that it can lead to premature convergence i.e. local minima. Elitism should be used with caution, if too many unaltered individuals are carried from one generation to another, it could lead to the loss of population diversity.

Once parents are selected, crossover operator combines them in a process of making offspring. As well as other operators, the crossover can be done in many different ways. A comparative study of many different crossover operators can be found in [19], [20], and [21]. The most popular solutions are single point crossover, multi-point crossover, uniform crossover, and others. Although all of these techniques are implemented, single point crossover produced the best results. This operator is one-point uniform: after selecting parent chromosomes $M(i,j)$ and $P(i,j)$, where $i \in (1,m)$, and $j \in (1,n)$. Length of the chromosome is n, and m is the number of matings. A cutting point C is randomly selected to be $C \in (2,n-1)$. Two offspring are then created by combining the first part of M chromosome to the second part of the P chromosome and vice versa. After crossover is done, chromosomes undergo a fixing process which is needed since single point crossover can make up to 50% of the population to be infeasible. A fixing consists of searching for the agent gene in the chromosome and changing its location to be the first gene of the chromosome. If the chromosomes starts with an agent gene, a fixing of this type is not needed.

Mutation, on the other hand, recovers some of the lost genetic material and tries to distribute genetic information randomly. An illustration of how mutation process affects chromosomes is shown in the Fig. 3. It also introduces new chromosome structures by randomly altering some of the genes. Six different mutation operators are introduced in [10], of which four do the gene altering in a similar way as it is done here. The mutations used in this algorithm are:

- Swap mutation; swaps the position of two task/agent identifiers (genes) in the chromosome
- Replace mutation; replaces one task/agent identifier with another from the list of possible tasks
- Shrink mutation is similar to replace mutation, and the only difference is that task/agent gene is replaced with the inactive gene.
- Growth mutation; replaces one inactive gene with an active task/agent gene.

The same mutation operators are used both for task genes and agent genes. It is noted that the algorithm yields better results if the mutation probability increases in the later stages of the algorithm. The reason for this is that over time population gets more homogenized, but usually, this leads to local minima or sub-optimal solution. This way the increased number of mutations can help producing better solutions. Mutation occurring probability is chosen to be rather small, from 1 to 10%. Setting mutation to 100% would give a completely random search process.

V. Fitness function

In this work a linear function that combines different variables and various user defined weights into a solution's cost. It is common for the best solution to have a cost equal to zero. Here that is not the case, and the minimum cost can differ as well as the overall feasibility of the candidate solution. In addition, it is necessary to select an optimization criterion. It is not sufficient for a solution to be only feasible, but the goal is to reach globally optimal solution in as many cases as possible. It cannot be guaranteed that the optimal solution will be found, since the algorithm always starts from a different position in a search space. Even if the initial population set is always the same, the algorithm would not output the same solution due to stochastic nature of genetic operators. In order to obtain a good solution,

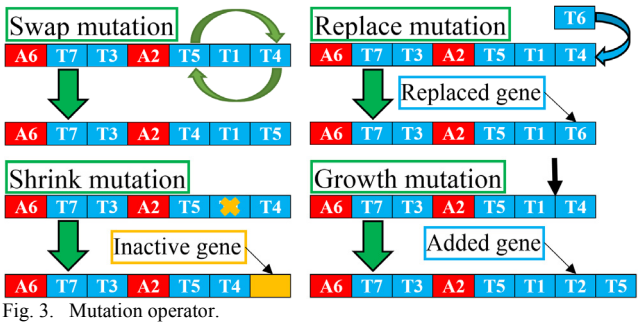

Fig. 3. Mutation operator.

the algorithm needs to run many times. A quality of the solution also depends on the optimization criterion, and in this work three different criteria can be selected:

- minimizing the makespan of the entire mission as shown in (2) (see below)
- minimizing the energy used for the whole mission as shown in (3) (see below)
- aggregating both time and energy, thus optimizing a multi-objective function (4)

Let T_j be the finish time of the last gene in an agent's plan vector $(T_1, T_2, \dots, T_j)$ and A_i is the makespan of i-th agent's plan.

$$A_i = (T_1, T_2, \dots, T_j) \tag{1}$$

Thus, the minimization function is as follows:

$$f(t) = \min(\max(\{A_1, \dots, A_n\}) + W_k \cdot \sum_{i=1}^{n} A_i) \tag{2}$$

where n is the number of agents used in a mission plan and W_k is a user defined weight.

Let E_n be the energy used per each agent in the mission, where n is the number of agents involved in the mission plan, then the energy minimization function will be:

$$f(e) = \min(\sum_{i=1}^{n} E_i) \tag{3}$$

$$f(x) = W_i \cdot f(t) + W_j \cdot f(e) \tag{4}$$

where W_i and W_j are weights defined by a user. Equations (2) and (3) should be normalized before $f(x)$ is calculated. Depending on the selected optimization criteria one of these two costs is selected or both. If mission duration is selected, the cost of the plan with longest makespan would be minimized. A weighted sum of agents' makespan is added in the (2) with a purpose to force minimization of each agent's makespan. Although, the first member of the equation is sufficient to minimize the mission duration, the second member minimizes the makespan of all agents. In this way some agents can be available for the use in some other mission, while some other agents still perform tasks in the first mission.

Two main levels can be distinguished in the fitness function. A "mission plan level" and an "agent plan level" (Fig 4). In the mission plan, all of the penalties, awards, and costs that are related to the entire mission are computed, while in the agent's plan level only task constraints and a single agent cost are calculated. There is also a sub level in the mission plan level that calculates agent's constraints and penalties.

The fitness function iterates through the population to, evaluate the fitness of each chromosome. A chromosome is firstly divided into agent plans. Each of these agent plans is tested and assessed separately, which means that each of these plans has its own penalty and makespan or energy cost that contribute to the fitness of the chromosome on the mission plan level. This procedure is repeated for the whole population in every generation. Penalties are calculated first, and then added to the cost function forming an overall fitness for a given plan. On mission plan level five different conflicts are sanctioned with a penalty. These penalties are added if:

- there are no agents in the mission plan
- one or more of MTs is missing from the plan
- more agents are used then there are available ones
- there is unassigned tasks or plan does not start with an agent gene
- there is an agent plan that does not have at least one of the MTs

In the mission plan level, beside penalties, there is also an award system. There are two types of awards, with the first one being awarded if the planner reaches a feasible solution, i.e. the sum of all penalties is equal to zero. The other award is being added for each agent used in the mission plan, thus favoring the use of as many agents as possible. This is especially important if the optimization criterion is the makespan of a mission. The

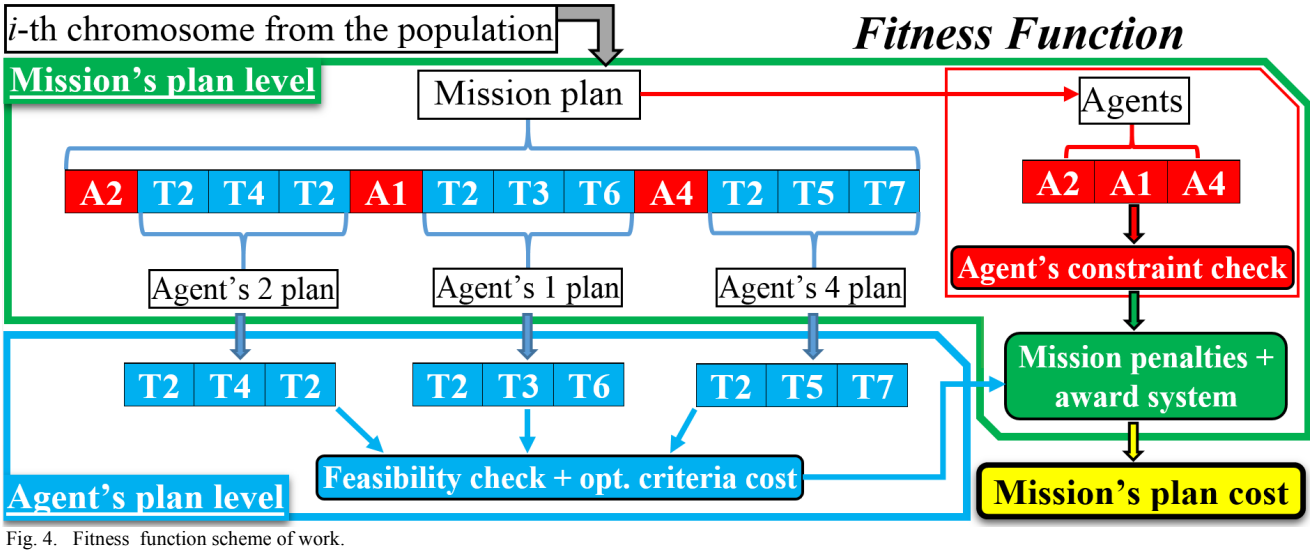

Fig. 4. Fitness function scheme of work.

agent plan level also has its penalties for the contempt of task constraints. These penalties are added if:

- preconditions are not fulfilled
- postconditions are not fulfilled
- location condition is not met for a given task

On the agent plan level, each plan is evaluated separately in the function that is in charge of plan evaluation. This function iterates through the plan and checks if task constraints are fulfilled. Besides the task constraints, this function calculates both the duration of the plan and the energy consumed by the agent to fulfill that plan. Both costs and penalties are summed and sent to the mission plan level where they are being added to the cost and penalties of the mission plan level. Now equation (4) is expanded into:

$$f(x) = W_i \cdot f(t) + W_j \cdot f(e) + P + D \qquad (5)$$

where P is the sum of all penalties and D is the sum of all awards in a mission plan. The sum of all penalties shows if the mission plan is feasible, i.e. if the sum is equal to zero, the mission is feasible, otherwise it is not. In general, feasible solutions are preferable even though they might not be optimal or even close to optimal solution. In the end all of these penalties, awards and costs are combined with weight factors and an overall mission plan cost is formed. This is the cost used, by selection operator for establishing the rank of chromosomes in the population set.

VI. Experimental Results

The presented genetic planner is implemented in MATLAB 2015b on a Windows workstation, i7@2.8GHz, and 32GB of RAM. It is usually challenging to find a good ratio between population size and a number of iterations executed. These two parameters are the one that can be set before running the GA that has the biggest influences on the running time. Running time is not the best indicator of the performance of GA since the load of CPU can vary. It is usually used to give a rough evaluation of GA's performance. A number of generations are used more often for evaluating the performance of GA. After many runs, it is noted that there is an almost linear correlation between time, the number of individuals, and the number of generations.

In the problem described in the case study section, for one iteration it takes an average of 0.003 seconds more per additional individual in the population set. Larger population size leads to a solution in a lesser number of generation, but as described the running time increases per generation. There is a need to find an optimal combination of these two parameters. In the Fig. 5 mean cost values for various population sizes are shown. GA was run 30 times with each set of parameters and then mean value was computed. It can be noted that the population size of 20 individuals is too small for this kind of problem, while the rest of the test, values have the same mean values at a higher number of iterations. At 500 iterations mean cost value is the same for the population size of 100 individuals, while at 800 iterations all values from 50-500 blend in at a minimum of mean cost value. The value of 500 iterations is chosen as the optimal one for stopping criterion, and for the population number 100 is chosen. Various population sizes are tested over 500 iterations run and presented in the Fig. 6. It can be noted that for

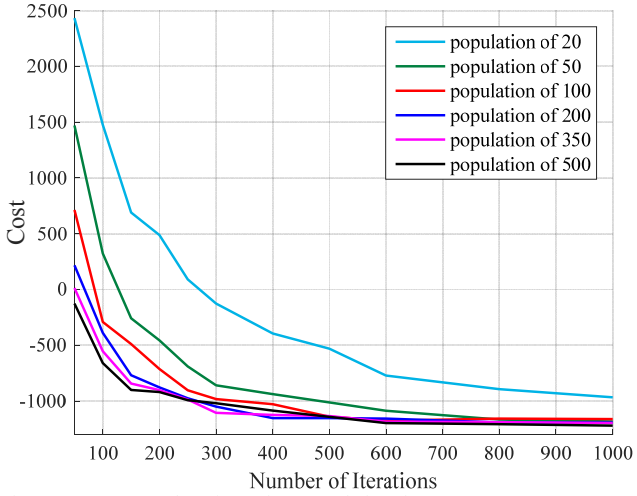

Fig. 5. Mean cost values for various population sizes.

populations over 400 iterations mean cost value and median values does not change much.

VII. Case Study

The mission scenario that will be presented here consists of six MTs. Three of these tasks are ScanArea (Task 3), and the other three are TakeImage (Task 4). In this scenario, mission planner has five agents at its disposal. Thus, at least one agent will have to perform two or more MTs. Every one of these agents has different velocity and energy consumption rate. Calculations of these parameters are fairly simplified since that is not in the scope of this paper. Simplified calculations are also used for evaluating tasks duration, i.e. the theoretical time that it takes for an agent to finish a task.

Although only six tasks are used as an input, the mission plan will end up having 29 active genes. As it is previously mentioned, inputs to the planner are so called MTs. MT can be any task from the list of tasks shown in Table 1. GA then makes sure that all necessary tasks for completion of MT are in the chromosome. A simple example of this is if Task 3 is chosen as MT and agent is not at the starting location defined in Task 3,

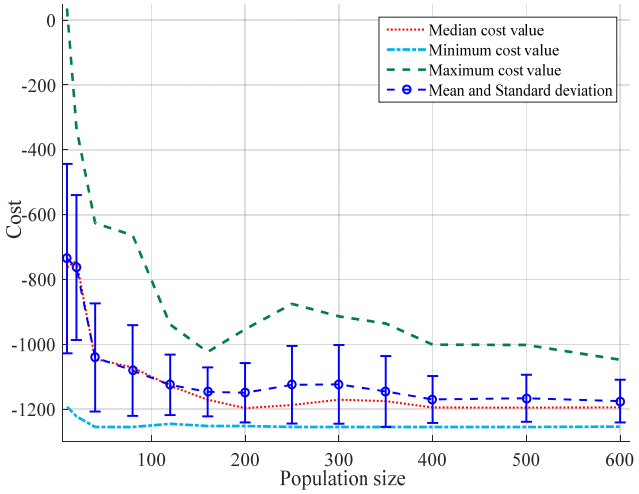

Fig. 6. Median, minimum, maximum, and mean cost values with standard deviation for various population sizes over 500 generations.

GA needs to generate Task 2 or Task 1 in the sequence before Task 3, otherwise, the fitness function will characterize this solution as infeasible.

The output of the genetic planner is a mission plan. X axis accounts for a timeline [s], while Y-axis represents tasks. Small battery icons in the top left corners show the energy level of each AUV after completion of assigned tasks. In this case, all tasks are assumed to be below the surface, and that is why there is no Task 1 in any of the plans. It can be noted that agents 1, 2, and 5 have some tasks running in parallel, thus making a makespan shorter. The mission plan for this particular scenario, represented by Gantt chart can be seen in Fig. 7. Parallel task constraints can be seen in Table 1.

Results for different population sizes run over 500 iterations are shown in Table 2. The table displays population size, mean value, standard deviation, median value, minimum and maximum values and the time took per run. Same columns except the first one, which now shows a number of generations, have Table 3. Results presented in Table 3. are for a various number of generations for a population size of 200.

Convergence rate of the genetic planner is shown in Fig 8. The blue curve stands for the minimum cost value in each generation, while magenta shows the average cost of the fittest individual throughout generations. Parameters used in this case study are as following: mutation rate 10%, the population size of 100, and the number of generations is limited to 500.

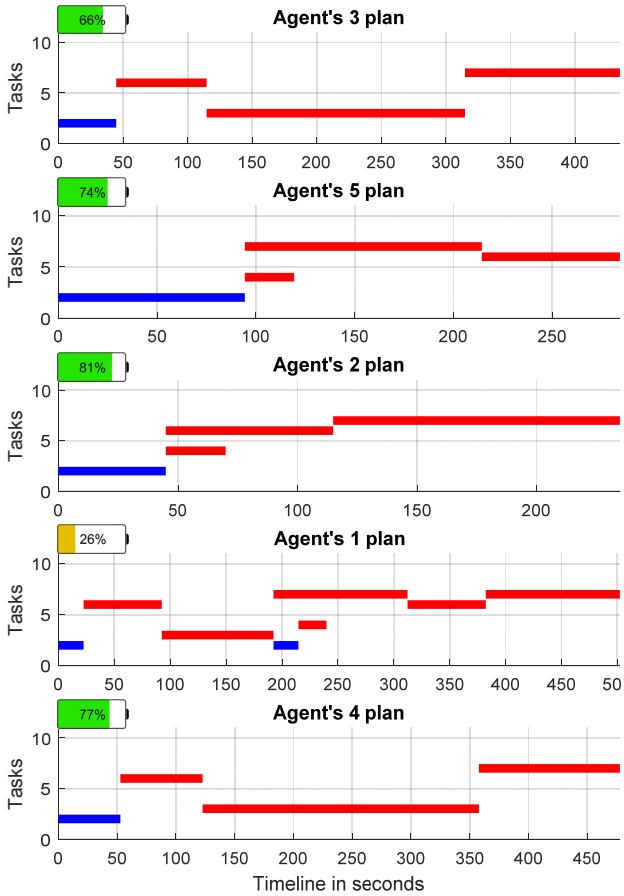

Fig. 7. Gantt Chart of a mission plan with battery status.

TABLE I. LIST OF AVAILABLE TASKS

Task #	Action	Parallel Task Constraints
Task 1	GoTo_aboveSurface	Task 4, Task 6, Task 7
Task 2	GoTo_belowSurface	Task 4, Task 6, Task 7
Task 3	ScanArea	Task 4, Task 5
Task 4	TakeImage	Task 1 & 2, Task 3, Task 6, Task 7
Task 5	Measure at Location	Task 4, Task 6, Task 7
Task 6	Comm. Status report	Task 1 & 2, Task 4, Task 5
Task 7	Comm. Send data	Task 1 & 2, Task 4, Task 5
Task 8	Progress monitoring	Task 1 & 2, Task 6, Task 7
Task 9	Follow Pipe	Task 4, Task 5, Task 6, Task 7
Task 10	Follow moving object	Task 4, Task 5, Task 6, Task 7
Task 11	"Dummy Task"	-

TABLE II. POPULATION SIZE RESULTS FOR 500 ITERATIONS

Pop.	Mean	Std.	Median	Min.	Max.	Time[a]
10	-735.4	292.3	-762.4	-1192.6	36.4	32.3
20	-763.1	223.6	-742.9	-1222.6	-332.1	62.7
40	-1040.7	166.8	-1045.5	-1255.1	-626.1	85.7
80	-1081.0	140.0	-1071.4	-1255.3	-663.6	143.8
120	-1125.2	93.2	-1128.0	-1245.4	-939.0	190.9
160	-1146.7	75.5	-1170.9	-1252.1	-1023.6	245.1
200	-1149.5	91.4	-1197.0	-1252.3	-953.6	295.6
250	-1125.0	119.7	-1187.4	-1255.1	-875.2	363.3
300	-1123.6	121.4	-1170.8	-1255.3	-913.9	425.3
350	-1145.5	109.2	-1175.3	-1255.3	-935.8	505.1
400	-1170.1	72.3	-1195.0	-1255.3	-1001.1	569.7
500	-1166.4	72.6	-1195.3	-1255.1	-1002.0	719.9
600	-1175.3	66.1	-1194.6	-1254.2	-1047.7	869.2

TABLE III. RESULTS FOR VARIOUS NUMBER OF GENERATIONS FOR 200 POPULATION SIZE

Gen.	Mean	Std.	Median	Min.	Max.	Time[a]
50	216.3	346.5	161.0	-274.7	891.4	27.6
100	-388.9	271.1	-433.2	-738.2	176.4	54.1
150	-768.4	226.2	-803.7	-1143.4	-172.1	80.5
200	-877.7	145.4	-868.3	-1155.6	-623.9	115.6
250	-975.7	102.5	-949.3	-1234.3	-811.2	131.3
300	-1049.5	96.9	-1033.0	-1255.3	-904.3	159.2
400	-1153.6	77.7	-1184.6	-1254.9	-1006.8	212.9
500	-1153.1	100.8	-1197.8	-1247.0	-864.8	263.5
600	-1157.0	88.6	-1198.9	-1255.3	-947.0	327.0
800	-1191.9	58.7	-1212.3	-1254.2	-1023.2	427.2
1000	-1193.9	56.2	-1208.6	-1254.2	-1041.1	544.4

[a] Time is expressed in seconds

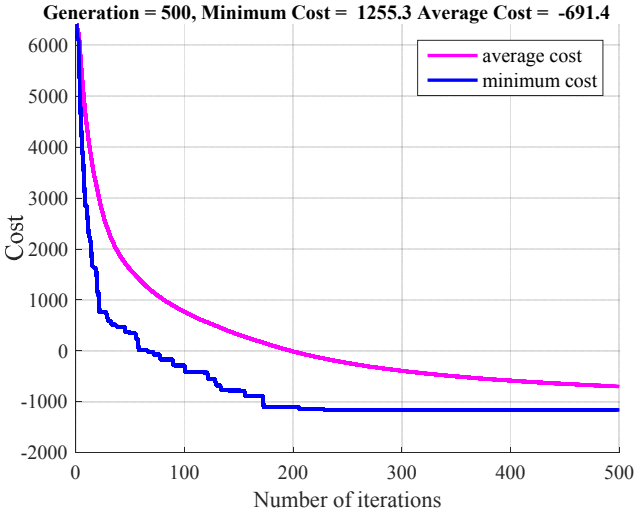

Fig. 8. Convergence plot.

VIII. Conclusion and Future Work

In this paper a genetic planner for multi-agent mission planning is presented. The novelty aspect of the proposed solution is that it can handle solutions that require task repetitions for a successful mission. The initial population set consists of random task/agent genes with variable chromosome length. Data structure used for encoding agents/tasks into a chromosome is based on integer identifiers, whereas a chromosome is represented as a vector.

As in other cases with population-based search, different parameters, and their initiated values, can greatly affect the performance of the proposed algorithm. However, for the problem presented in Section VII. Case Study, approxametaly 500 iterations and population size of 100 - 200 individuals give the best results in a reasonable time span. For the future work it is planned to extend the current system in the following directions:

- include an option for handling multi-agent tasks to allow assignement of several agents to a task. This also includes adding a "delay" gene in the chromosome representation.
- extend the fitness function for multi-objective search, in order to deal with two or more (e.g. makespan, energy consumtion, and mission safety/risk) criterion forming a Pareto optimal front.
- introduce more heterogeneity in an AUV set, especially in the sensors domain. This will also result in additional test in the cases of using AUVs from different manufactorers.
- propose a complete Mission Management Tool (MMT) for computing the initial planning of activities, monitor their execution and in the case of unexpected events make appropriate decisions about re-planning.

Acknowledgment

The research leading to the presented results has been undertaken within the SWARMs European project (Smart and Networking Underwater Robots in Cooperation Meshes), under Grant Agreement n. 662107-SWARMs-ECSEL-2014-1, which is partially supported by the ECSEL JU and the VINNOVA.

References

[1] M. Castilho, L. A. Kunzle, E. Lecheta, V. Palodeto and F. Silva, "An investigation on genetic algorithms for generic STRIPS planning," in *Advances in Artificial Intelligence - IBERAMIA*, Berlin, Springer-Verlag, 2004, pp. 185-194.

[2] R. E. Fikes and N. J. Nilsson, "STRIPS: a new approach to the application of theorem proving to problem solving," in *International Joint Conference on Artificial Intelligence*, London, England, 1971.

[3] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld and D. Wilkins, "PDDL-the planning domain definition language," Yale Center for Computational Vision and Control, New Haven, 1998.

[4] J. H. Holland, Adaptation in natural and artificial systems, MIT Press, 1975.

[5] B. P. Gerky and J. M. Maja, "A formal analysis and taxonomy of task allocation in multi-robot systems," *Internation Journal of Robotics Research,* vol. 23, no. 9, pp. 939-954, 2004.

[6] A. Boukhtouta, A. Bedrouni, J. Berger, F. Bouak and A. Guitouni, "A survey of military planning systems," in *The 9th ICCRTS Int. Command and Control Research and Technology Symposium*, Copenhagen, Denmark, 2004.

[7] W. Hongjian and B. Xinqian, "A GA Path Planner Based on Domain Knowledge for AUV," *Oceans'04,* pp. 1570-1573, 2004.

[8] X. C. Ding, A. Pinto and A. Surana, "Strategic planning under uncertainties via constrained markov decision processes," in *International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013.

[9] L. Belfares, W. Klibi, N. Lo and Guitouni, "Multi-objective tabu search based algorithm for progressive resource allocation," *European Journal of Operational Research,* no. 177, pp. 1779-1799, 2007.

[10] A. H. Brie and P. Morignot, "Genetic planning using variable length chormosomes," in *International Conference on Automated Planning and Scheduling*, Monterey, 2005.

[11] M. Khouadija, M. Schoenauer, V. Vidal, J. Dreo and P. Saveant, "Pareto-Based multiobjective AI planning," in *23rd International Joint Conference on Artificial Intelligence*, Beijing, China, 2013.

[12] C. Ramirez-Atencia, G. Bello-Orgaz, M. R-Moreno and D. Camacho, "A hybrid MOGA-CSP for multi-UAV mission planning," in *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*, Madrid, Spain, 2015.

[13] G. Bello-Orgaz, C. Ramirez-Atencia, J. Fradera-Gil and D. Camacho, "GAMPP: Genetic algorithm for UAV mission planning problems," *Intelligent Distributed Computing IX,* pp. 167-176, 2016.

[14] L. Geng, Y. Zhang, J. Wang, J. Y. Fuh and S. Teo, "Cooperative task planning for multiple autonomous uavs with graph representation and genetic algorithm," in *Control and Automation (ICCA)*, Hangzhou, China, 2013.

[15] J. Magalhaes-Mendes, "Project scheduling using a competitive genetic algorithm," in *International Conference on Simulation, Modelling and Optimization*, Santander, 2008.

[16] Vanhoucke, D. Debels and Mario, "A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem," *Operations Research,* vol. 55, no. 3, pp. 457-469, 2007.

[17] L. García-Hernández, A. Araúzo-Azofra and L. Salas-Morera, "A review on encoding schemes used by genetic algorithms in plant layout design," in *13th International conference on project engineering*, Badajoz, 2009.

[18] Z. Michalewicz, Genetic algorithms + data structures = evolution programs, Springer-Verlag Berlin Heidelberg, 1996.

[19] C. H. Westerberg and J. Levine, "Genplan: combining genetic programming and planning," in *19th Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG)*, Milton Keynes, England, 2000.

[20] T. Blickle and L. Thiele, "A comparison of selection schemes used in evolutionary algorithms," *Journal of Evolutionary Computation,* vol. 4, no. 4, pp. 361-394, 1996.

[21] J. Magalhaes-Mendes, "A comparative study of crossover operators for genetic algorithms to solve the job shop scheduling problem," *WSEAS Transaction on Computers,* vol. 12, no. 4, pp. 164-173, 2013.

[22] N. Soni and T. Kumar, "Study of various crossover operators in genetic algorithms," *International Journal of Computer Science and Information Technologies,* vol. 5, pp. 7235-7238, 2014.

[23] S. Picek, M. Golub and D. Jakobovic, "Evaluation of crossover operator performance in genetic algorithms with binary representation," in *Bio-Inspired Computing and Applications*, Berlin, Springer Berlin Heidelberg, 2011, pp. 223-230.