Robotica: (2020) volume 38, pp. 512–530. © Cambridge University Press 2019
doi:10.1017/S026357471900081X

Robot Mission Planning using Co-evolutionary Optimization

Rahul Kala*

Robotics and Machine Intelligence Laboratory, Indian Institute of Information Technology, Allahabad, India

(Accepted April 24, 2019. First published online: June 4, 2019)

SUMMARY
Mission planning is a complex motion planning problem specified by using Temporal Logic constituting of Boolean and temporal operators, typically solved by model verification algorithms with an exponential complexity. The paper proposes co-evolutionary optimization thus building an iterative solution to the problem. The language for mission specification is generic enough to represent everyday missions, while specific enough to design heuristics. The mission is broken into components which cooperate with each other. The experiments confirm that the robot is able to outperform the search, evolutionary and model verification techniques. The results are demonstrated by using a Pioneer LX robot.

KEYWORDS: Evolutionary robotics; Robot motion planning; Mission planning; Cooperative co-evolution; Genetic algorithm.

1. Introduction

Imagine working in a modern futuristic office that provides a robot instead of a human peon to carry out all the clerical jobs. A typical instruction you would give to the robot is to: *Deliver five papers to five different people, get an approval from any of your project investigators and then to get the consent of the head of the department, to take attendance of students operating at different laboratories, meanwhile to also check for office mail, and simultaneously to also get you some stationery from the office after the request for stationery has been approved by the administrator and the list of requirements has been gathered from all the sub-ordinates*. The other workers in the office may have similar requirements given to the robot via a web interface. The robot must be able to plan its sequence of actions so as to cater to all the requirements, while optimizing the total span of time it takes to complete all the tasks. This problem is called as the mission planning problem, while the complex task given to the robot is called as the mission. Classic robotic systems believed that any robotic problem can be decomposed into going to the site of interest, doing some manipulation and coming back. This is called as the classical problem of robot motion planning which is to find a solution to the query "*Go from A to B avoiding obstacles*".[1,2] The mentioned unit tasks in the problem involve interaction and manipulation along with mobility, which is also now a rather solved problem.

The context of application is service robotics, wherein the robots are required to render a number of services in a home or office environment. The instructions given to the robots are complex expressions written in a formal language, representing the problem and the constraints of the problem. The robot must solve for the problem statement and get a sequence of steps to be performed in order to fully solve the problem. This constitutes a higher level problem that is expected to be solved by the robot and is called as the problem of mission planning.

Any general mission is a specification using Temporal Logic,[3,4] which consists of Boolean operators (and, or, not) and temporal operations (next, eventually, globally, until). The given Linear

* Corresponding author. E-mail: rkala001@gmail.com

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

Temporal Logic (LTL) formula can be converted into an automaton. The transition system specifies the feasibility of transiting from one state to the other. The problem is hence to find a string representing a solution, which is accepted by the automaton representing the temporal logic and the specified transition system. The approach is widely used, however suffers the problem of complexity. The conversion of the temporal logic into an automaton has an exponential complexity with respect to the number of propositional variables. Further, most solutions do not cater to optimality.

Currently, the model verification techniques are widely used for mission planning, the futuristic scenario will see far too many users that will explode the number of proposition variables and render the model verification techniques as useless. In this paper, the approach is hence to use evolutionary computation for the problem of mission planning. The evolutionary computation techniques are probabilistically optimal and probabilistically complete. They are hence good candidates to solve problems with a very large number of variables.

Evolutionary computation has found its application in a large number of planning and scheduling problems. The current problems being solved by using evolutionary computation are generalizations of the generic Travelling Salesman Problem and do not have temporal constraints. The generic heuristics and methods used do not hold good for the case of temporal constraints. The aim here is to hence use evolutionary computation for the aspects of scalability, probabilistic optimality and probabilistic completeness; while to also enable a better generalization of the problem by allowing more realistic and complex problems to be specified by using a formal language.

Evolutionary computation cannot simply replace the search operation to solve the full class of problems that can be represented by LTL, because the development of a system for verification of the solution is still exponential in complexity. Hence, a new language is proposed to represent the mission. The language designed should have the characteristics of representing a wide variety of missions of everyday life, while being verifiable in a polynomial time. Further, the language should have some structure that can be exploited as heuristics by the search technique. In this paper, a language is proposed that ensures all the characteristics.

Cooperative evolution[5,6] is a very useful technique to evolve good components that make up the complete problem. Each component has its own evolution system undergoing selection, crossover, mutation and other evolutionary operators. However, the fitness of a component is not assigned based on its own individual goodness, but based on how well does it cooperate with the other components to best solve the particular problem. A related problem is competitive evolution wherein the different solutions compete with each other by assignment of a fitness relative to the performance of the solution when competing with the other solutions. The different solutions therefore encourage each other to improve themselves by developing good characteristics.

The main contributions of the paper are: (i) A new language is proposed for the specification of missions inspired by real-life scenarios. (ii) A co-evolutionary approach to mission planning is proposed which is scalable to a very large number of mission sights, while being probabilistically optimal and probabilistically complete. The co-evolutionary optimization attacks the problem structure for the design. (iii) Structure of the problem is exploited to design heuristics for good initial population generation and evolution. (iv) Results are shown over very large problem sizes, unsolvable by all known solvers to similar problems. The proposed approach easily surpasses search-based approaches, model verification-based approaches and other evolutionary approaches. Overall, the use of evolution for such a problem is a very new approach to the best of the knowledge of the author. (v) Results are demonstrated on the Pioneer LX robot using ROS framework.

2. Related Works

To the best of the knowledge of the author, the paper is the first attempt to solve such complex missions using evolutionary computation and therefore there is no direct related work. However, it is necessary to understand the independent advances of the two communities of model verification-based mission planning and evolutionary planning, so as to appreciate the proposed work.

The well-adopted mechanism of solving the problem is by the use of model verification techniques with an exponential complexity. Kress-Gazit et al.[7] used LTL specification as a mission to generate a controller for the robot in both discrete and continuous domains, while also extending the work to the multi-robot scenarios. In another work, Lahijanian et al.[8] incorporated the use of probabilities to

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

compute a mission plan only if all components of the plan adhere to the probabilistic safety specified by the user. The LTL language was so generalized to incorporate probabilities. The transition probabilities were learnt by using Reinforcement Learning. While many papers of the domain do not account for optimality, the work of Svorenova et al.[9] modelled preferences and rewards so as to always focus the search towards the states satisfying the goal criterion, while also maximizing for the rewards. Lyons et al.[10] further introduced uncertainties and made a framework for mission verification with multiple robots. While the current results in this cluster are very encouraging, the approaches can never scale up to the level of future demands due to the exponential complexity involved.

Efforts are however made to generate quality solution in small computation times. Bhatia et al.[11] used sampling for the low-level planning. They also showed the phenomenon of exploration of the transition system to provide more information to make the plans at the higher level so as to satisfy the mission specification converted into an automaton. The work was also extended12 to the use of hybrid nonlinear dynamics and a lazy evaluation of the high-level plans by the low-level planner. In another similar work, McMahon and Plaku[13] proposed the use of heuristics to guide the expansion of the search tree. The aim was to get a solution early, while to also get a solution of low cost. Jeong et al.[14] proposed the use of Q-RRT* algorithm that quickly finds a solution and then converges to the optima. The algorithm was used for directing robots operating using a Developmental Episodic Memory-based Mechanism of Thought implemented using Adaptive Resonance Theory-based network. Heuristics are also used in solving problems of task planning and geometric planning in an integrated manner,[15] while task planning closely represents missions. Some overall mechanisms of task planning and interactions with low-level planners are shown in the work of Maraghy and Rondeau.[16]

The other community is planning using evolutionary computation, for which the most relevant approaches come in the use of cooperative co-evolution. However, the approaches have only solved the problem of multi-robot motion planning and not the problem of mission planning involving the robots to visit a set of mission sites, which have Boolean and temporal constraints. Wang and Wu[17] presented such a framework wherein multiple robots were planned in a co-evolutionary framework, with the inter-robot collisions or collisions with any obstacles handled by penalization. Each robot's path denoted a component, while representatives of a component were used for fitness evaluation of the other component. Similarly, Dongyong et al.[18] presented another co-evolutionary framework, while using multi-objective criterions for fitness assignment. Kala[19] used Grammatical Evolution while co-evolving paths for multiple robots. A memory-based unit stored optimal paths between node pairs, which was used to speed up optimization. The robots moved in a maze-like structure and had to cooperate so as not to block each other on the way. In all these contexts, the robot solves for a classic motion planning problem. Zhang et al.[20] solved the problem of Air Traffic Management in which the aircrafts were dynamically grouped and an optimal trajectory was computed by co-evolution. Berenson et al.[21] solved the problem of robot grasping in which three populations were maintained for source configuration, goal configuration and grasps. The populations cooperated with each other to optimize the grasping points which were connected using RRT as planning algorithm. Similarly, Ćurković and Jerbić[22] modelled the two arms of a robot as populations that cooperated with each other for planning. The solutions are not generalizable to represent mission planning.

However, using evolutionary computation, some other works solve for missions. In all cases, the missions are very specific and mostly representing a Travelling Salesman Problem. A generic mission planning representing Boolean and temporal constraints is yet to be solved by any evolutionary technique. Xidias et al.[23] solved for a group of robotic vehicles visiting a set of robotic sights using a centralized genetic algorithm. The centralized approach solved for all objectives and constraints simultaneously. The approach was also extended to incorporate constraints and the use of Bump surfaces.[24] Kala[25] used an optimization-based approach to avoid the collision between the robots. The initial planning was done independent of the robots, while the mutual collisions were resolved using optimization. Similarly, Xiao et al.[26] presented an evolutionary framework for the generic problem of robot motion planning. In a different sense, Shirazi and Jin[27] used Delaunay triangulation for global motion planning and a heuristic approach for local motion planning for motion planning of a swarm of robots. Kala[28] solved the problem wherein a team of robots was asked to compulsorily visit a set of mission sights. From an evolution perspective, the problem was equivalent to a combinatorial optimization (optimize the visit order of the robots) and a parametric optimization (to assign robots

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

to each mission sight). Some temporal constraints in optimization are also shown in ref. [29,30], which however represent constraints as a geometric (or continuous variable) and not at the task or mission level.

In another related work, Tsiogas and Lane[31] solved a similar mission planning of Correlated Team Orienteering, wherein a collection of robots was supposed to visit a number of sites, while maximizing the reward along with some user-specified timing criterions. The authors optimized the sequences, while also satisfying the resource constraints. Lamont et al.[32] first solved the hybrid problem of route planning and path planning of an aerial vehicle. Thereafter, the trajectory was used for the motion of a swarm of robotic vehicles. Similarly, Geng et al.[33] also made an aerial vehicle solve the complete coverage problem by visiting all possible sites. The authors first optimized the route and then the robotic trajectory. Yu and Cai[34] took the mission of robot exploration, wherein a robot was supposed to explore and visit a number of sites. The authors used Heterogeneous Interactive Cultural Hybrid Algorithm, wherein the bounding ceiling spaces were used along with a user interface for a better performance. Baalbaki et al.[35] optimized for assignment and scheduling problem associated with the multi-robot missions, wherein a type of mission could only be given to some types of robots. Again, the robots had to collectively solve for all possible tasks in the mission. All such missions and problem modelling represent a Travelling Salesman Problem constrained by kinematics only, and the generics of Boolean and Temporal Constraints cannot be easily fed in.

3. Language for Mission Specification

The language is inspired by the everyday requirements of home and office. The language defines a *sub-task* as a logical construct of mission sights or places to visit (which in a classical sense serve as a propositional variable). This makes it possible to state sub-tasks such as visit any three sights, any two of a group of sights, etc. The *task* is defined as a temporal ordering of sub-tasks, which makes it possible to order the robot to do something followed by something else, and so on. A *mission* is a collection of tasks that can be solved in parallel. Consider the example: *Get an approval from any of your project investigators and then to get the consent of the head of department, to take attendance of students operating at different laboratories*. Here the sub-tasks "*get an approval from any of your project investigators*" and then "*get the consent of the head of department*" make a task which must be performed one after the other, which together with the other task (of only one sub-task) "*take attendance of students operating at different laboratories*" make a mission. Complex English statements can be easily represented by simple logical constructs making a sub-task.

It may sometimes appear that the optimal solution of a task is simply appending the optimal solution to sub-tasks one after the other. However, the starting location of a sub-task depends upon the ending location of the previous sub-task, and hence it may be useful to sub-optimally solve a sub-task so as to end the sub-task at a preferable place for the start of the next sub-task. As in the above example, delivering paper to some Mr. *X* at the end may be preferable if his office is close to the head of department, while a typical Travelling Salesman solution may not call this strategy as optimal. Similarly, for multiple tasks, solving one task after the other may not be optimal as two tasks may have related sub-tasks. So, taking attendance from the Robotics Lab just after visiting Mr. *X* who sits close to the Robotics Lab is suggestive.

A *sub-task* (φ) is simply a propositional logic of mission sights, given by the grammar (1).

$$\begin{aligned} \varphi &::= \sigma \\ \sigma &::= \sigma \wedge \sigma | \sigma \vee \sigma | \urcorner \sigma | (\sigma) \\ \sigma &::= \Diamond\sigma_1 | \Diamond\sigma_2 | \Diamond\sigma_3 | \Diamond\sigma_4 | \ldots . \Diamond\sigma_n \end{aligned} \tag{1}$$

Here $\wedge$ is the 'and' operator, $\vee$ is the 'or' operator, $\urcorner$ is the 'not' operator, $\Diamond$ is the 'eventually' operator from Linear Temporal Logic (LTL), n is the number of mission sights.

The verification whether a string s satisfies the sub-task given by grammar (1) is simple. First, the truthness of the terminals is asserted, given by Eq. (2). Here, $s(t)$ represents the tth character of the string.

$$s| = \Diamond\sigma_i \text{ iff } \exists t : s\ (t) = \sigma_i \tag{2}$$

This is done by making a hash table and marking all the characters in string s as true with a complexity of $O(|\sigma| + |s|)$, where $|\sigma|$ is the maximum size of a sub-task and $|s|$ is the maximum

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

length of the string. Replacing all terminals with true or false gives a Boolean expression which can be easily solved. The expression is first converted into postfix, and then evaluated.

A *task* (ψ) is an ordering of the sub-task wherein all specified sub-tasks must be performed in the same order, given using the sequencing formulation of LTL, given by Eq. (3). The task can be understood by the word "then" with the task ψ specifying perform φ_1 then φ_2 then φ_3 and so on.

$$\psi = \Diamond(\phi_1 \wedge \Diamond(\phi_2 \wedge \Diamond(\phi_3 \wedge ...))) \tag{3}$$

Similarly, the verification of a string s against a task needs to be ascertained. Let φ_i be the ith sub-task in the task ψ and let υ_i be the first index in the string at which the sub-task φ_i is entailed. υ_i may be calculated by Eqs. (4) and (5). Here, $s(i:j)$ denotes the sub-string from index i to j.

$$\upsilon_1 = \begin{cases} \min_j \ (j : s\ (1:j)\ | = \phi_1) & \text{if s}| = \phi_1 \\ \infty & otherwise \end{cases} \tag{4}$$

$$\upsilon_i = \begin{cases} & if\, \upsilon_{i-1} \neq \infty, \\ \min_j (j : s\ (\upsilon_{i-1} + 1 : j)\ | = \phi_i) & s\ (\upsilon_{i-1} + 1 : end)\ | = \phi_i, \\ & i > 1 \\ \infty & otherwise \end{cases} \tag{5}$$

Here, s consisting of sub-tasks $\varphi_1, \varphi_2, \varphi_3 \ldots \varphi_m$ entails ψ if $\upsilon_m \neq \infty$. Here, m is the number of sub-tasks in the task. The algorithm has a complexity of $O(|\sigma|.|s| + |s|^2)$, since the sub-task verification algorithm is called for all sub-strings whose size is under $|s|$, for a maximum of $|s|$ times.

A *mission* (χ) is defined as a collection of tasks, which must all be performed in any order and any part of a task or sub-task may be performed anywhere in between another task or sub-task. The mission is given by Eq. (6). The mission χ can be read as perform task ψ_1 and ψ_2 and ψ_3 and so on, in any order.

$$\chi = \Diamond\psi_1 \wedge \Diamond\psi_2 \wedge ....\Diamond\psi_p \tag{6}$$

Here, p is the number of tasks in the mission. A string s entails the mission simply if it entails every task specified in the mission given by Eq. (7).

$$s| = \Diamond\chi \text{ iff } s| = \psi_i \forall i \tag{7}$$

Correspondingly, the complexity of a mission verification system is $O(|\psi|.|\sigma|.|s| + |\psi||s|^2)$, where $|\psi|$ is the maximum number of tasks.

4. Evolutionary Search

4.1. Design philosophy

The problem is to generate a string that satisfies the mission. This can be naively solved by a genetic algorithm. However, a very strong heuristic exists in the structure by which the language has been designed. The different sub-tasks form sub-components that are only weakly cross-dependent on each other. Consider that you are required to take signatures from any two of the three colleagues (A, B and C) and then from any of the three deans (D, E or F). At first sight, it appears that the problem can be decomposed into two independent problems of first taking signatures from any two of the three colleagues, and to use the last colleague as the source of the new problem of taking signature of any dean. However, if C sits closer to a Dean, it may be better to take his/her signature at the end, in which case a Dean's signature will be very easy to take. This does not mean that the problem is actually completely dependent. If either of these sub-tasks were very complex, it was better to solve them independently to get some good solution to sub-tasks and then to see how do they integrate to form a good solution to the task. Co-evolution is hence used, wherein each sub-task forms a component, which has its own evolution process including the initial population generation and genetic operators.

The mission consists of a number of tasks, all of which must be solved. The simplest strategy is to solve all tasks one by one, thereby completing the mission, which is of course far from optimal. The problem is more complicated as any sub-task in a task may be solved in between any other

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

task of the same mission. Further many sub-tasks may be redundant, in which case solving one task may automatically solve a few others and no further attempt must be made. The co-evolutionary framework on one hand helps in identifying structures that can be separated, however on the other hand makes it difficult to check for these constraints.

The solution operates in a two-layer hierarchy, the slaves representing the co-evolutionary sub-task solvers, and a master genetic algorithm representing the mission solver. The slave co-evolutionary algorithm is used to generate good solutions to the sub-task. The sub-tasks together make a solution of the task and thereafter the mission. Therefore, the sub-tasks do not have a fitness component of themselves. However, the co-evolution style of working can be used for the generation of good solutions which combine with other good sub-task solutions to generate an overall good mission solution. Even though a naïve implementation of co-evolution could be used for getting good sub-task solutions, here the overall evolution of the sub-task using co-evolution is coordinated by a master genetic algorithm. In principality, the master genetic algorithm fuses different sub-tasks together to make a solution to a task, and the different task solutions to make a solution to a mission. The master genetic algorithm is responsible for fusing the different sub-components produced by the slave genetic algorithms evolving good solutions of sub-tasks into a single linear path that solves all sub-tasks and tasks. The master genetic algorithm takes care of the characteristics noted above. The two algorithms are naturally linked. The master genetic algorithm, as a population, has pointers to different individuals of the slaves and the master genetic algorithm does not have its own strings as population, since its only role is the fusion of slave solutions. The slave genetic algorithm relies upon the master genetic algorithm for fitness evaluation. The master genetic algorithm indicates the good solutions that can be formulated by a slave solution and the contribution of a slave solution in this effect. This is used by the slave genetic algorithm for fitness computation.

4.2. Slave co-evolutionary sub-task solver

First, the easier component is discussed, which is the evolution of the sub-components in cooperation with each other. This section discusses the complete evolution except for fitness computation. The solution of a sub-task is a classic combinatorial optimization problem with the difference that not all the mission sights as mentioned in the sub-task may be visited. On the contrary, only some of these may be visited, whose number is unknown. Consider asking to visit any two of three colleagues, the optimal length of the individual is two and not more. Hence the genetic individual is variable in size and correspondingly the different operators are modified. Assume $var(\varphi)$ denotes the variables which appear in sub-task φ, excluding the variables which appear as a negation. The random individual generated is a concatenation of random permutations of all variables of the sub-task.

The crossover operator is modified to account for the variable size given by Algorithm 1. The basic motivation is to keep the count of mission sights in the first child same as the count of mission sight in the first parent, and similarly with the second child. The combinatorial version first constructs a map of the count of all the mission sights encountered as genes in both the parents. First half of both the parents are given directly to the children, while the second half of the parents is given to the

Algorithm 1: CrossoverSlave(ψ, φ,X,Y)

$C_1 \leftarrow \emptyset$, $C_2 \leftarrow \emptyset$
for i from 1 to len(X)/2, add X(i) to C_1
for i from 1 to len(Y)/2, add $Y(i)$ to C_2
for i from len(X)/2 + 1 to len(X)
  if X(i) $\notin C_2$, add X(i) to C_2
for i from len(Y)/2 + 1 to len(Y)
  if $Y(i) \notin C_1$, add $Y(i)$ to C_1
while size(C_1)<size(X)
  $\sigma_r \leftarrow$ random var(φ)
  if $\sigma_r \notin C_1$, add σ_r to C_1
while size(C_2)<size(Y)
  $\sigma_r \leftarrow$ random var(φ)
  if $\sigma_r \notin C_2$, add σ_r to C_2
return C_1, C_2

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

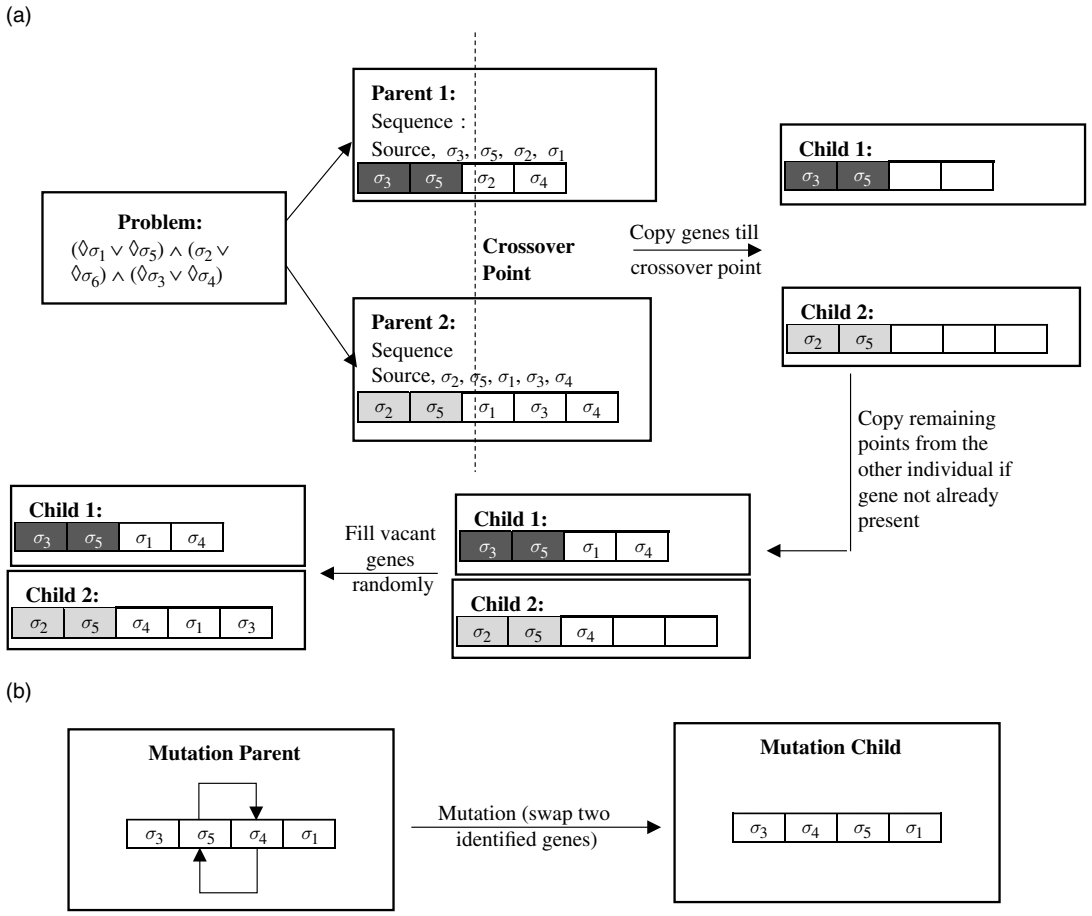


Fig. 1. Slave Genetic Framework. (a) Crossover. In crossover, the shaded genes (half) are directly taken from the parents. The others are inserted from the other parent, only if the gene is not already present. So σ_3 could not be inserted from individual 2 to child 1 and σ_2 could not be inserted from individual 1 to child 2. The vacant genes are randomly filled. (b) Mutation. Two random genes are swapped in the parent to create the child.

opposite child as long as the count of the mission sights does not exceed the count in the parent. To match count of mission sights, the missing mission sights in the children as compared to the parents are added in a random permutation.

The mutation operator simply swaps random genes. The algorithm uses crossover and mutation to create the child population pool, which together with the parent population pool competes for existence as the population of the next generation. Stochastic Universal Sampling[36] with rank-based expectation[37,38] is used for selection. One of the little programming challenges involved is to take the mission specification as text, which would be suited for the user, and to parse the string to identify tasks, sub-tasks and variables and to store them as relevant data structures. The problem statement consists of a text with every task given in a new-line. Each line is first processed to extract all sub-tasks. The task string is continuously broken around the "then" operator, until all sub-tasks are extracted out. The identification of all variables in the sub-task text is done by a regular expression that extract tokens between operators. In order to check whether a string satisfies the sub-task, similarly, all variables are replaced by the ground truth available from the string. The string is then converted into a postfix expression and solved. A typical individual, crossover and mutation operators are explained in Fig. 1.

4.3. Master genetic algorithm

Co-evolution only generates good components and leaves the task of selection of the components in order to make a full qualified solution to the complete problem, which is another optimization problem. Here, a master Genetic Algorithm is used that continuously hunts for better solutions by

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

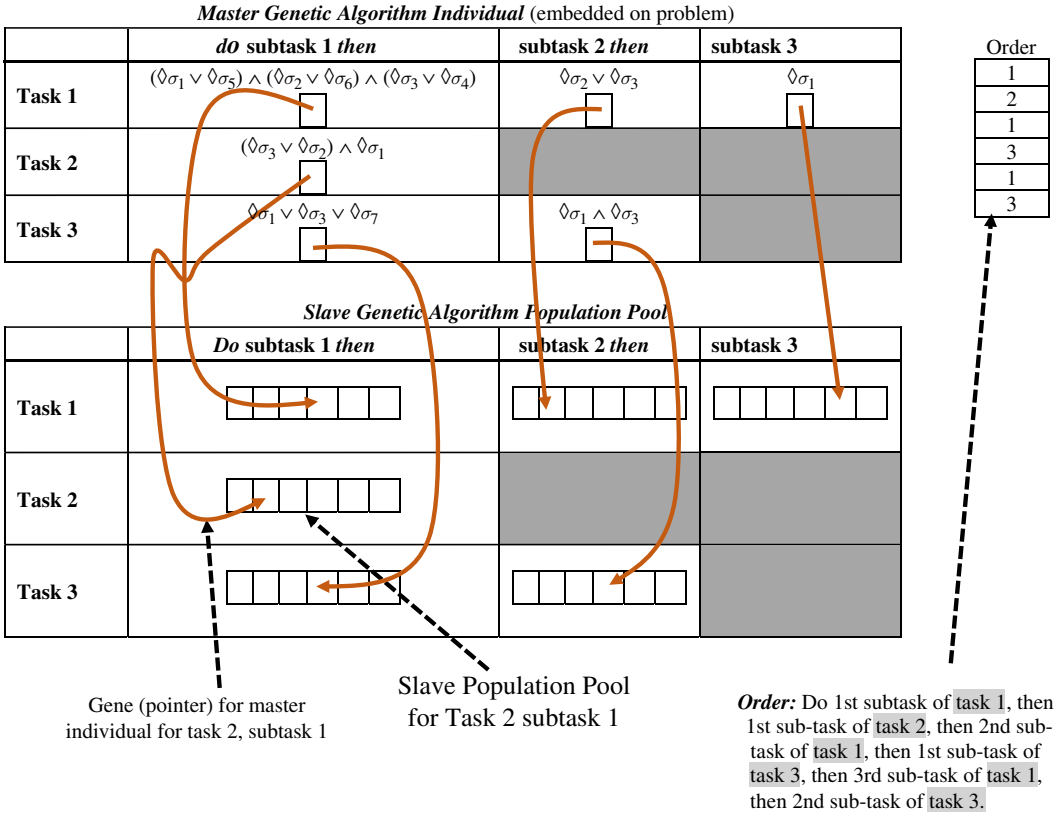


Fig. 2. Co-evolutionary Mission Solver. The table at the bottom of the figure represents the slave population, with a population pool (shown in every cell) for every sub-task inside a task and an individual representation as given in Fig. 1. Each cell of individual (shown in the table at the top of the figure) is a pointer to a random individual in the corresponding slave genetic algorithm population. Order is a sequence of highlighted text.

selecting the sub-components from a slave co-evolutionary setup. Hence two evolutions run in parallel, a co-evolutionary framework that generates good diverse solutions to the different sub-tasks, and a master genetic algorithm that selects the solutions for sub-tasks in order to make the solution of a task and mission.

The major design issue in the master is the design of an individual. The design is presented in Fig. 2. Let φ_i^j denote the ith sub-task of task j. The generalized mission specification is given by Eq. (8).

$$\chi = \Diamond\psi_1 \wedge \Diamond\psi_2 \wedge ....\Diamond\psi_p \tag{8}$$

$$\psi_i = \Diamond\left(\phi_1^i \wedge \Diamond\left(\phi_2^i \wedge \Diamond\left(\phi_3^i \wedge ...\right)\right)\right)$$

The master genetic algorithm holds a solution from the entire population for every sub-task in representation. The master genetic algorithm also holds additional genes specifying the order in which the sub-tasks must be solved. The order genes are added in addition to the choice of individuals specifying the solutions to the sub-tasks. Hence, let S denote the solutions of the sub-tasks taken from the slave population and O denote the encoded order of sub-tasks, as two components of the master population individual.

Let O denote the encoded order of sub-tasks of a problem in order to solve a mission. The sub-tasks have a temporal relation in a task, meaning that one sub-task φ_i^j cannot be performed out of sequence and has to be executed only after sub-task $\varphi_{i-1}{}^j$ has been completed. Hence the encoding of operation order of sub-tasks only contains the tasks. At time t, the task $O(t)$ is executed, while the specific sub-task inside $O(t)$ to be executed is the one as per sequence. Let $Z(t)$ denote the sub-task that will be performed at order sequence t, a mapping from O to Z is given by Eqs. (9) and (10).

$$Z(t) = \varphi_i^{O(t)} \tag{9}$$

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

$$i = \begin{cases} count\ (O(t-j) = O(t), 1 \leq j \leq t-1)\ t > 1 \\ 1 \qquad\qquad\qquad\qquad\qquad\qquad\quad t = 1 \end{cases} \tag{10}$$

A constraint that must be satisfied for feasible setting of the encoding O, is that the size of O should be the total number of sub-tasks contained in the mission and every task should have exactly the same number of entries as the number of tasks that it has, given by Eq. (11).

$$size(O) = \sum_{i=1}^{p} size\ (\psi_i),\ count\ (O(t) = \psi_i) = sublen(\psi_i) \tag{11}$$

Here, $sublen(\psi_i)$ is the function that returns the number of sub tasks in the task ψ_i. The master genetic algorithm is a simple genetic algorithm, whose initial population is generated by copying random individuals from the different populations of the slave co-evolutionary framework. In addition, the order is initialized, which is a random string satisfying constraint (11). The crossover operator used is the uniform crossover technique,[39] wherein randomly approximately half the solutions to sub-tasks are taken from the first parent and the other half from the other parent. The crossover of order variables needs to be taken care of as a conventional combinatorial optimization problem with the exception that variables will repeat, the count however must be constant. The technique is given as Algorithm 2. The operator is also explained by Fig. 3.

Algorithm 2: CrossoverMaster(X,Y)

//Part I: For the solution to sub-task
X$S \leftarrow$ S(X), Y$S \leftarrow$ S(Y)
C$S_1 \leftarrow \emptyset$, C$S_2 \leftarrow \emptyset$
for i from 1 to len(XS)
  with 0.5 probability, add XS_i to CS_1, add YS_i to CS_2
  with 0.5 probability, add YS_i to CS_1, add XS_i to CS_2
//Part II: For order
X$O \leftarrow$ O(X), Y$O \leftarrow$ O(Y)
C$O_1 \leftarrow \emptyset$, C$O_2 \leftarrow \emptyset$
M$_1$(x) $\leftarrow$ count of x in XO M$_2$(x) $\leftarrow$ count of x in YO
for i from 1 to len(XO)/2,
  add XO(i) to CO_1, M$_1$(XO(i))$\leftarrow$ M$_1$(XO(i))-1
for i from 1 to len(YO)/2,
  add YO(i) to CO_2, M$_2$(YO(i))$\leftarrow$ M$_2$(YO(i))-1
for i from len(XO)/2 + 1 to len(XO)
  if M$_1$(XO(i))>0,
    add XO(i) to CO_2, M$_1$(XO(i))$\leftarrow$ M$_1$(XO(i))-1
for i from len(YO)/2 + 1 to len(YO)
  if M$_2$(YO(i))>0,
    add YO(i) to CO_1, M$_2$(YO(i))$\leftarrow$ M$_2$(YO(i))-1
L$_1 \leftarrow \emptyset$, L$_2 \leftarrow \emptyset$
for all i in M$_1$
  while M$_1$(i)>0, add i to L$_1$, M$_1$(i)$\leftarrow$ M$_1$(i)-1
for all i in M$_2$
  while M$_2$(i)>0, add i to L$_2$, M$_2$(i)$\leftarrow$ M$_2$(i)-1
add random permutation of L$_1$ to CO$_1$
add random permutation of L$_2$ to CO$_2$
C$_1 \leftarrow <$ CS_1, CO_1 $>$, C$_2 \leftarrow <$ CS_2, CO_2 $>$
return C$_1$, C$_2$

The mutation operator simply selects a different individual for one sub-task, sampled from the individuals in the sub-task's population pool. For the order component, the mutation is a simple swap of two genes given by Algorithm 3. The operator is also described in Fig. 4.

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

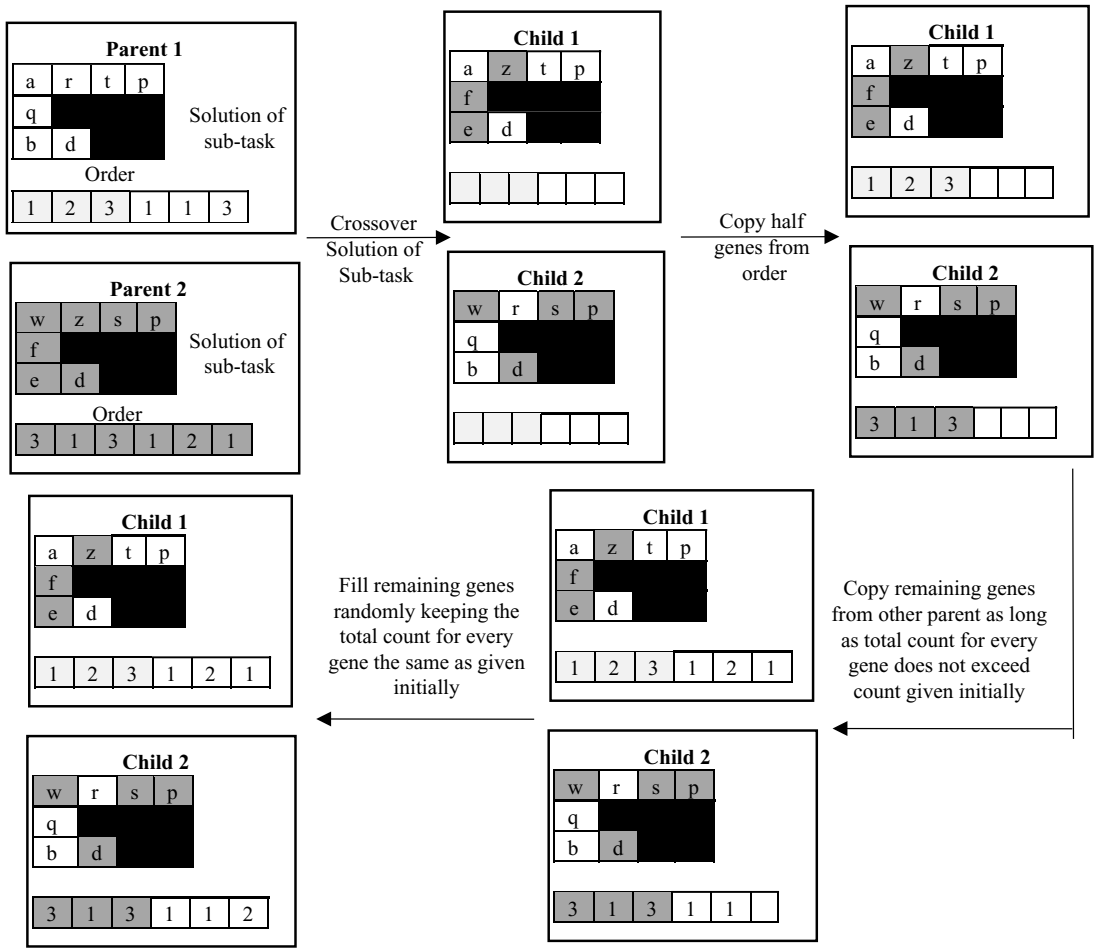


Fig. 3. Crossover Operator. The letters represent pointers to some individual in slave population as shown in Fig. 2. The scattered crossover technique is used. For order, half the genes are directly transferred from the parent, while the other half only if the number of occurrences in the child does not exceed the same in the parent. So 3 from parent 1 could not be inserted in child 2. The remaining genes are inserted as a random permutation of left genes.

Algorithm 3: MutateMaster(X)

//Part I: For the solution to sub-task
X$S \leftarrow S$(X), $r_1 \leftarrow$ random gene
$p \leftarrow$ random individual from PopulationSlave$_{r1}$
X$S_{r1} \leftarrow p$
//Part II: For order
X$O \leftarrow O$(X)
$r_1 \leftarrow$ random gene in XO, $r_2 \leftarrow$ random gene in XO
swap(X(r_1), X(r_2))
return <XS, XO>

4.4. Fitness assignment

One of the most challenging tasks in the framework is the fitness assignment to the individuals, which guides the evolution. This is done differently for both the master genetic algorithm and the slave co-evolutionary algorithm. For the master genetic algorithm, the fitness evaluation is simply the cost (distance) to complete all the sub-tasks, with a penalty proportional to the number of sub-tasks that could not be completed due to infeasibility. However, care must be given that computation of fitness shall solve for each sub-task only from the genes taken from that sub-task's slave population. Hence

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

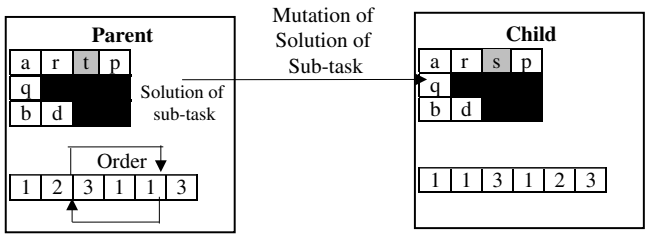


Fig. 4. Mutation Operator. For any cell, the pointer is changed to a different slave individual, and for order any two genes are swapped.

concatenation of all sub-strings into one string and using the fitness evaluation function detailed previously is not a solution, as it will reward a slave individual if it also partly solves the next sub-task in sequence, which may initially look good, but will hamper the structure of the problem and hence the utility of co-evolution. The fitness function of master is given as Algorithm 4.

Algorithm 4: FitnessMaster(X)

if contains(H,X), return H(X)
X$S \leftarrow S$(X), X$O \leftarrow O$(X)
counter(ψ) $\leftarrow 1 \forall \psi$
$\pi(\psi) \leftarrow \emptyset \forall \psi$
cost $\leftarrow 0$, last $\leftarrow 0$, penalty $\leftarrow 0$
for t from 1 to size(SO)
  $\psi \leftarrow O$(t), $\varphi \leftarrow \varphi^{O(t)}_{counter(O(t))}$, solved $\leftarrow$ false
  for i from 1 to size(SX_φ)
    cost$\leftarrow$ cost+d(last, X$S_\varphi(i)$), last$\leftarrow$ X$S_\varphi(i)$
    for all task ψ_t,
      if X$S_\varphi(i)_{\notin}\pi(\psi_1)$, add X$S_\varphi(i)$ to $\pi(\psi_t)$
    if satisfies($SX_\varphi(1{:}i)$, φ),
      solved $\leftarrow$ true, break
  if not solved, penalty $\leftarrow$ penalty+1
  $\pi(\psi) \leftarrow \emptyset$, counter($\psi$)$\leftarrow$ counter(ψ)+1
return cost+α.penalty

The fitness function computation is a simple implementation of (4) and (5). The only addition is that a penalty is added to the cost proportional to the number of sub-tasks not fulfilled by the planner. Further, once a sub-string of the individual satisfies a mission specification, all subsequent sub-strings are trimmed off as they are clearly unnecessarily increasing the cost of the solution. The algorithm is given by Algorithm 4. Since the fitness computation is very complex, a hash table (H) is used to store the fitness of all individuals. If an individual happens to be repeated, its fitness value is given directly from the hash table.

For the slave population, the fitness assignment needs to assign a very good fitness if it appears to be a very good solution of the sub-component as well as if it helps the other sub-components to be very good, even though itself not being the best. Two metrics are used for fitness assignment for the slave population. The first is substitution, wherein the best k individuals are taken from the master genetic algorithm, and the sub-component in the same master population is replaced with the one whose fitness is being evaluated. The change in fitness value due to this substitution, averaged over best k individuals is measured. The second component of the slave fitness function comes from the individual merit. The algorithm for the slave fitness function is given as Algorithm 5.

The slave population has a major responsibility of generation of good solution components and hence the slave evolution is given a memetic framework by inserting a few iterations of a local optimization algorithm. The fitness function for the slave is very computationally expensive, and therefore not usable for even small number of iterations of the memetic framework. The aim in local

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

Algorithm 5: FitnessSlave(ψ, φ,X)

$s_1 \leftarrow 0$
for Y in best k master population individuals
  $Z \leftarrow Y$, $Z_{\psi,\varphi} \leftarrow X$
  $s_1 \leftarrow s_1 + (\text{fitnessMaster}(Z) - \text{fitnessMaster}(Y))$
$s_1 \leftarrow s_1/k$
if not satisfies (X, φ), penalty $\leftarrow 1$,
else penalty $\leftarrow 0$
return $s_1 + \beta\|X\| + \alpha$.penalty

optimization is not to add to the complexity of the algorithm. Each individual of the slave represents a solution to a sub-task, and during local optimization a non-cooperative measure consisting of the individual's length (plus a heavy penalty for not solving the mission is used) is used which is computationally less expensive. If the individual so generated makes the individual worse when judged for cooperation, the individual will not survive the selection and will die. This is especially very useful in the initial generations when the population is random and good sub-task solutions can speed up convergence.

4.5. Overall algorithm

The overall algorithm is given as Algorithm 6. In the algorithm, *per* is the random permutation generator, [] denotes an ordered list, ',' is taken as the list append operator, $[[\psi_1]_{\text{sublen}(\psi 1)}, [\psi_1]_{\text{sublen}(\psi 2)}, \ldots .[\psi_p]_{\text{sublen}(\psi_p)}]]$ denote an ordered list formed by concatenation of ψ_1 for $sublen(\psi_1)$ times, ψ_2 for $sublen(\psi_2)$ times, and so on.

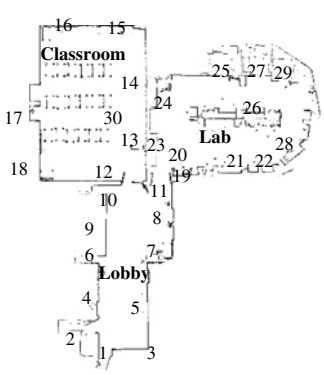
5. Experimental Setup

This section presents the additional modules and the complete experimental setup. The experiments are done using the Pioneer LX robot on a ROS framework. The Pioneer LX robot was operated

Algorithm 6: Co-evolutionary Task Solver

Parse Mission Text
//Initialize Slave Population
for every task ψ, For every subtask φ in ψ
  for j from 1 to slave population size
    $\text{PopSlave}_{\psi,\varphi}{}^{j} \leftarrow$ random_permutation(var(φ))
//Initialize Master Population
for i from 1 to master population size,
  for every task ψ, for every sub task φ in ψ,
    $S_\psi, \varphi \leftarrow$ random individual from $\text{PopSlave}_\psi, \varphi$
$O \leftarrow \text{per}([[\psi_1]_{\text{sublen}(\psi 1)}, [\psi_1]_{\text{sublen}(\psi 2),\ldots.[\psi_p]_{\text{sublen}(\psi p)}}])$
$\text{PopMaster}^i \leftarrow <S, O>$
Calculate Fitness of all individuals in master and slave
While stopping criterion is not true
//Master Genetic Algorithm
  CrossoverMaster
  MutationMaster
  Fitness evaluation master
  Selection Master
  For every task ψ, For every subtask φ in ψ
    //Slave Genetic Algorithm
    CrossoverSlave(ψ, φ)
    MutationSlave(ψ, φ)
    LocalOptimization(ψ, φ)
    SelectionSlave(ψ, φ)
    Fitness Evaluation Slave (ψ, φ)

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press



(1) Office GCN
(2) Office RK
(3) Robita entrance gate
(4) Office Mary
(5) Waiting Area
(6) Cabin Joey
(7) Copying Station
(8) Office PC
(9) Cabin Ross
(10) Classroom Entrance
(11) Robita lab entrance
(12) Light switches
(13) Classroom System
(14) Whiteboard
(15) Wall AC front
(16) Wall AC Back
(17) Back Right AC
(18) Back Left AC
(19) Charging Station
(20) Robot Home
(21) Racheal
(22) Monica
(23) Left Cupboard
(24) Right Cupboard
(25) Divya
(26) Latika
(27) Priyanshu
(28) Heena
(29) Bishnu
30) Projector

Fig. 5. Map along with Mission Sites.

using the ARNL libraries. The first step was to make a map of the robot, which is facilitated by the presence of a lidar sensor and high-resolution encoders present in the robot. A SLAM algorithm does the task of creating a map. The different sites of interest like classroom, offices, etc. were annotated during the map creation. The different goals are edited using the Mapper3 software, which allows to give descriptive names of the goals and to extract their coordinates. The robot needs to be initially localized manually, if it does not start from home or the last position left. Thereafter, the robot is able to continuously localize itself. The location can be easily computed by the robot using the high-resolution wheel encoders along with the lidar scan used to map the readings with the known map. The robot uses a mixture of deliberative and reactive planning in order to move. The planning happens by finding a collision-free trajectory connecting the current position to the goal. The onboard lidar senses any new obstacle and change of map, which is used to make a local map for navigation. The goal is specified by using ROS messages given to the ARNL library.

The evolutionary algorithm designed needs input as the transition cost between all mission sites. For the same, Probabilistic Roadmap[40] technique was used to convert the workspace into a roadmap, the mission sites were added as additional vertices. The specific implementation was adapted from an earlier approach.[41] The roadmap so produced was used by a Floyd–Warshall algorithm to compute the transitions between all mission sites. A Human Robot Interaction module was developed for the robot that spoke the task to be performed to let the human understand. The robot planned its path from any position to the immediate goal using planning on the static map initially built, over which the sensor readings were mapped and reactively handled by the robot.

The area that could be tested was limited to three rooms only because of presence of stairs and doors that the robot cannot negotiate. This mechanism to test the robot was extremely painstaking and hence a synthetic map was created wherein many mission sites were randomly added. Further, a utility was made to generate and visualize randomly generated missions. Testing was also done using this simulation setup.

6. Results

6.1. Hardware results

The map produced by the robot is given in Fig. 5. The map also shows the mission sites. In order to save space, only one result is being discussed.

- **Task 1:** A requisition needs to be signed by any one of the students and verified by any other student from the same set for the usage of an equipment. The set of users consist of Divya (25), Latika (26), Priyanshu (27), Heena (28), Bishnu (29), Mary (29) and Joey (6). The requisition must then be photocopied for records at the copying station (7). The photocopy must then be returned to any of the users from the same set of people.
- **Task 2:** Simultaneously a class needs to be prepared for which a marker is needed from the left cupboard (23) and a duster from the right cupboard (24). Since it is a pleasant weather, only two ACs out of the set Wall AC front (15), Wall AC back (16), Back right AC (17) and Back left AC

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

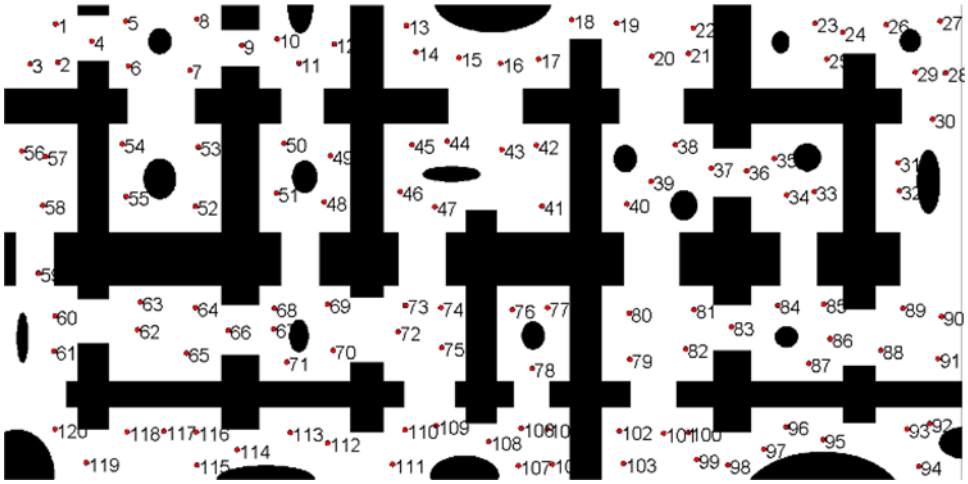


Fig. 6. Roadmap and mission sights.

(18) must be switched on, along with light switches (12) and classroom system (13). After the classroom is prepared, inform the secretary Mary (4).

- **Task 3:** At the same time a student was involved in disciplinary action and a notice should be given. The papers will be signed by GCN (1) and RK (1). A printout of evidence must be taken from office PC(8). The notice must then be given to any one of the culprits, Racheal (21) or Monica (22).

The results are supplied as a video at ref. [42]. The sequence taken by the robot is (i) Go to (6) Joey for signature, (ii) Go to Mary (4) for verification, (iii) Go to copying station (7) to take a photocopy, (iv) Go to Joey (6) to return the documents, (v) Go to Left Cupboard (23) to take marker, (vi) Go to Right Cupboard (24) to take duster, (vii) Go to Classroom System (13) and turn it on, (viii) Go to Wall AC Front (15) to switch on the AC, (ix) Go to Back left AC (18) to switch on the AC, (x) Go to Light Switches (12) to turn on the lights, (xi) Go to Mary (4) to report classroom ready work completion, (xii) Go to (2) RK to take a signature, (xiii) Go to GCN (1) to take a signature, (xiv) Go to Office PC (8) to take documentary evidence, and (xv) Go to Racheal (21) to serve disciplinary notice.

6.2. Simulation results

The testing is also done over a very long synthetic scenario of a map representing a very big office scenario with a resolution of 2000 × 1000. In order to test the algorithm, a very complex scenario with 120 mission sights was taken, shown in Fig. 6.

A random mission generator was used using the operators $\wedge$ with a probability of 0.7, $\vee$ with a probability of 0.1 and sequencing operator with a probability of 0.2. The consecutive $\vee$ was limited to 4. The sequence was generated as a "and of or" to minimize the simplification by the $\vee$ operator. The operands were random permutation of all the mission sights and 30% of the mission sights randomly replicated. An abuse of notation is used by skipping the eventual operator, representing sub-tasks in separate lines with '*t*' denoting the sequencing operator.

In order to save space, the simulations results are only displayed as a video,[43] which is self-explanatory. The sub-tasks are listed one after the other and the tasks are listed at different places. It may appear that the robot travelled long distances, while a Travelling Salesman Problem solution would have been much smaller. This is exactly the problem with temporal constraints wherein every sub-task is a small Travelling Salesman Problem and hence the robot is expected to make around 102 traversals, equivalent to the number of sub-tasks. The size considering all the sub-tasks, tasks and repeated variables was 620, which is extremely large

6.3. Comparisons

In order to analyse the algorithm, six problems are randomly generated. The problem sizes are of sizes 50, 75, 100, 125 and 150, considering only the distinct variables. The actual size is with a

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

30% repetition of variables for sub-tasks and repeated over 2–5 tasks depending upon the size of the problem. The stopping criterions are 4 min for the problems of sizes 50 and 75; and 30, 40 and 50 min for the problems of sizes 100, 125 and 150, respectively.

The first choice of comparison was the Uniform Cost Search which is an optimal graph search algorithm. The search failed after a problem size of 12 due to a shortage of memory. In contrast, the proposed algorithm test cases started with a size of 50. This is a significant difference which can be analytically explained by the exponential computation time of the Uniform Cost Search. Hence Iterative Lengthening algorithm was used, a linear memory variant. Again, the algorithm started taking too much time on a small increase in problem size, and the algorithm could not solve problems of size larger than 12 for prolonged time. The exponential computation complexity suggests the same results theoretically as well.

The mechanism adopted therein was to leave optimality as an objective and go for formal verification tools. NuSMV[44] was used to solve the problem. The tool did well while not being optimal. However, again the curse of dimensionality became effective and the tool could not solve problems of sizes more than 23 even after leaving it for over an hour.

A more practical comparison was using a basic evolutionary technique without any heuristics as a baseline method. Here, the fitness evaluation function was kept as a blackbox. Any custom optimization solver does not have the advantage of representing population of variable sizes, hence a variable individual sized-genetic algorithm was coded for the purpose. For small population sizes, it appeared that the method will be effective, however when the problem sizes increased to 25, the minimum size in the approach, it was not possible to get even a single random individual as feasible. The evolutionary pressures were towards reducing cost and not reducing infeasibility (which required an increase in cost). Hence, the solutions kept becoming worse and the technique had to be discarded.

By experiments and using theoretical analysis both, it is evident no solver can solve problem of such high size. Hence in order to get some comparison, a new evolutionary algorithm was proposed that represented most of the features of the problem structure and thus attacked the problem structure to be used as heuristics to the maximum extent possible. The solution does not completely represent a combinatorial optimization problem, however has most characteristics of combinatorial optimization. To exploit this structure both combinatorial and parametric optimization variants of operators were used. In order to generate an initial feasible solution, random permutations of the variables appearing in a sub-task were appended to make an initial feasible solution for a task; and thereafter multiple task solutions were added in a random order to make a feasible solution of a mission. A few solutions with random permutation of all variables and a few randomly generated solutions of maximum size were also added. The crossover was similar to the proposed approach while the mutation operator considered mission sight insertion, mission sight deletion, mission site swapping and replacement of a random mission site with another site. The solution hence most closely represents the proposed solution with the absence of co-evolution. The fitness function is repeated for all tasks and the maximum time taken to execute any task is taken as the fitness function.

The results are given in Fig. 7. The results are averaged for 40 independent runs and the standard deviation is shown by error bars. It is eminent that the proposed approach gives earlier convergence and converges to better values for large problems as compared to a basic evolutionary approach modified to the maximum extent to exploit problem structure. Further statistical testing is done for every problem at the mid-point of optimization and at the end of the optimization. For problem size 50, the baseline performs better, however for all other problem sizes the proposed approach performs significantly better than the baseline. The only exception is problem size 75 at the end of the optimization, in which case neither of the algorithms is statistically significant to the other.

6.4. Analysis

Next some analysis is done specific to the proposed algorithm to get more insights into these. The results are plotted in Fig. 8. The convergence along with generations for problem sizes is shown in Fig. 8(a). The computation time is large when the individuals are infeasible and large in size, which gets reduced in the later generations when the individuals are feasible and have been trimmed. The computation time along with increasing generations is shown in Fig. 8(b). The analysis did not show such a strong trend. The computation time largely comes from fitness evaluation that has a complexity of $O(|\psi|.|\sigma|.|s| + |\psi||s|^2)$. The plot of computation time per generation with problem size is shown in Fig. 8(c), which shows the same trends. Finally, to understand how difficult optimization is when

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

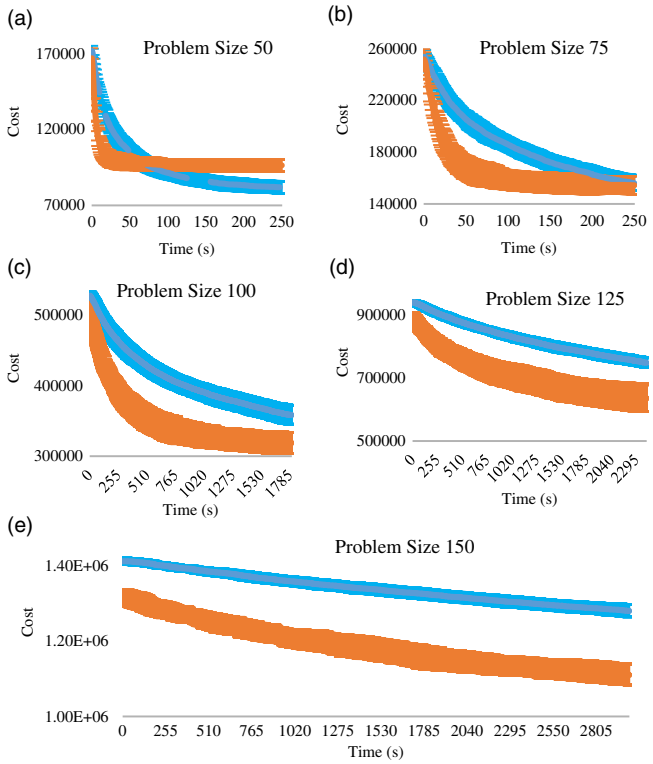


Fig. 7. Comparative Convergence of the algorithms for mission solving.

the problem size increases, the cost per unit problem size is plotted against problem size for different times. Dividing by problem size gives some basis of analysis since larger problems will have larger costs. The results are shown in Fig. 8(d). The increase in costs along with increasing time suggests the increase in difficulty, while the reduction in costs with increasing time is due to convergence.

The algorithm performance is also tested for the stability and performance with respect to the parameter values. The first parameter is k, the number of individuals against which the co-evolution checks for performance measure of an individual. The higher value of the parameter is a better indicator of cooperation, but at the same time takes a lot longer time to compute. The trend for values of 1, 2 and 3 is shown in Fig. 9(a), plotting the mean and the standard deviation as error bar. It can be seen that the performance is stable with respect to the parameter values, while higher values show a slower convergence. The other factor is β, the relative contribution of the individual merit of a slave co-evolution individual in contrast to the fitness considering the other individuals in a cooperative sense. The trend is shown in Fig. 9(b). Initially, the experiments were done on small realistic values for which the performance was stable. To test the performance very high values of β are also tested. This obviously results in convergence to a local optimal. Lastly, local search was used to make the algorithm in a memetic framework. The local search was designed so as to take very little time while operating on an easy to compute fitness function, or the length of the individual. Hence, no major trend is seen for different values of the factor which supports the design. The results are shown in Fig. 9(c). The figure does not show the error bars for simplicity.

7. Conclusions

This paper presents an approach for very complex mission planning using evolutionary technique where the mission sights have temporal constraints and numerous such mission specifications are given, to the best of the knowledge of the author. The future expectation from robots is to solve very challenging missions, which appears outside the computational capabilities of the search-based techniques. The paper presents a co-evolutionary approach for the same. The approach was far better than any known technique for the specific language and outperformed a baseline evolutionary planning technique for very large-sized inputs. The greatest advantage of co-evolutionary computation is its iterative nature, guaranteeing the availability of solution at any time, even with very large input

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

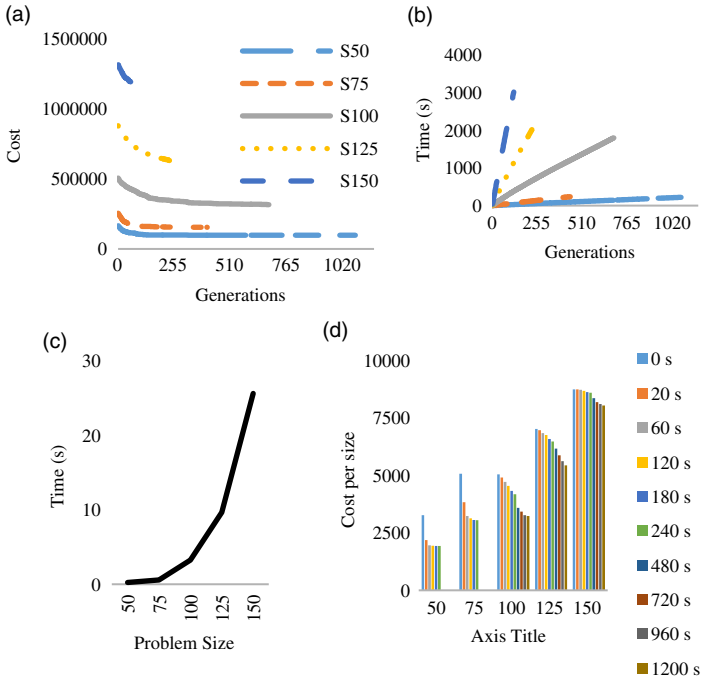


Fig. 8. Analysis of the co-evolutionary mission solver (a) Convergence (b) Computation time along with generations (c) Computation time per generation (d) Difficulty in optimization with increasing problem sizes.

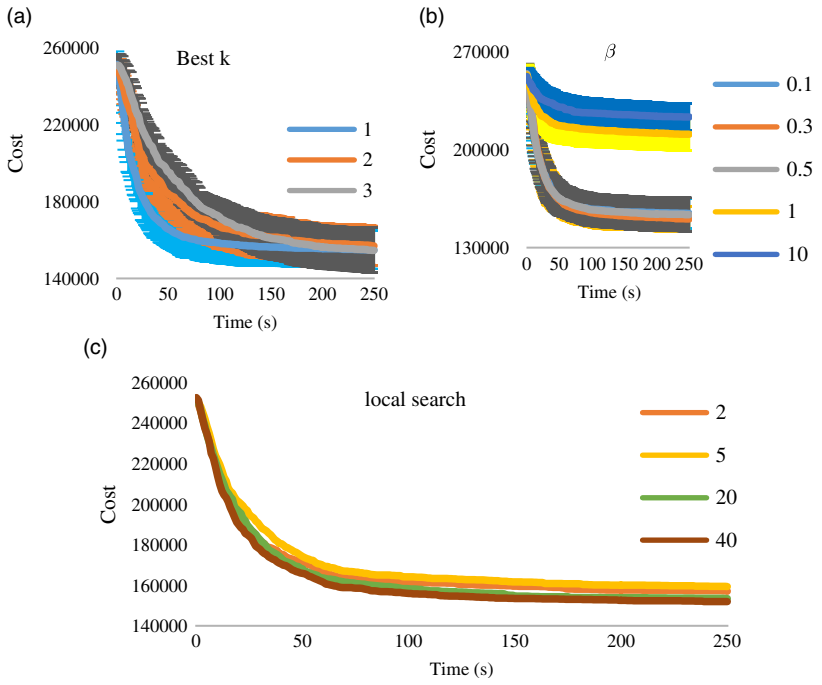


Fig. 9. Parameter analysis (a) The number of best components used in co-evolution, (b) β (c) Number of iterations of local search.

problems which forms an ideal characteristic. Overall, the paper presents a promising evolutionary solution to a new domain of mission planning.

There are however a few considerations. The humans do not necessarily have minimization of the total working time as the objective. Humans may consider earlier completion of a task or sub-task even if that comes with lengthening the solution path. Further, the possibilities of heuristics for the problem better needs to be explored. The complex missions are usually distributed between robots

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

and the missions usually have uncertain information which is available from the sensors only at the execution time. All this need to be worked out in the future.

Acknowledgements
The work is supported by the Science and Engineering Research Board, Department of Science and Technology, Government of India through grant number ECR/2015/000406.

References

1. H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations* (MIT Press, Cambridge, MA, 2005).
2. R. Tiwari, A. Shukla and R. Kala, *Intelligent Planning for Mobile Robotics: Algorithmic Approaches* (IGI Global Publishers, Hershey, PA, 2013).
3. C. Baier and J. P. Katoen, *Principles of Model Checking* (MIT Press, Cambridge, MA, 2008).
4. M. Fisher, *An Introduction to Practical Formal Methods Using Temporal Logic* (Wiley, West Sussex, UK, 2011).
5. M. A. Potter and K. A. DeJong, "A cooperative coevolutionary approach to function optimization," *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, Jerusalem, Israel (Springer-Verlag, Berlin, Germany, 1994) pp. 249–257.
6. M. A. Potter and K. A. De Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evolutionary Comput.* **8**(1), 1–29 (2000).
7. H. Kress-Gazit, G. E. Fainekos and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Trans. Robotics* **25**(6), 1370–1381 (2009).
8. M. Lahijanian, S. B. Andersson and C. Belta, "Temporal logic motion planning and control with probabilistic satisfaction guarantees," *IEEE Trans. Robotics* **28**(2), 396–409 (2012).
9. M. Svorenova, J. Tumova, J. Barnat and I. Cerna, "Attraction-based receding horizon path planning with temporal logic constraints," *Proceedings of the 2012 IEEE 51st Annual Conference on Decision and Control*, Maui, Hawaii (2012) pp. 6749–6754.
10. D. M. Lyons, R. C. Arkin, S. Jiang, M. O'Brien, F. Tang and P. Tang, "Performance verification for robot missions in uncertain environments," *Robotics Autonomous Syst.* **98**, 89–104 (2017).
11. A. Bhatia, L. E. Kavraki and M. Y. Vardi, "Sampling-based motion planning with temporal goals," *Proceedings of the 2010 IEEE International Conference on Robotics and Automation*, Anchorage, AK (2010) pp. 2689–2696.
12. A. Bhatia, L. E. Kavraki and M. Y. Vardi, "Motion planning with hybrid dynamics and temporal goals," *Proceedings of the 49th IEEE Conference on Decision and Control*, Atlanta, Georgia (2010) pp. 1108–1115.
13. J. McMahon and E. Plaku, "Sampling-based tree search with discrete abstractions for motion planning with dynamics and temporal logic," *Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, IL (2014) pp. 3726–3733.
14. I. B. Jeong, W. R. Ko, G. M. Park, D. H. Kim, Y. H. Yoo and J. H. Kim, "Task intelligence of robots: Neural model-based mechanism of thought and online motion planning," *IEEE Trans. Emerging Topics Comput. Intell.* **1**(1), 41–50 (2017).
15. P. Munoz, M. D. R-Moreno and D. F. Barrero, "Unified framework for path-planning and task-planning for autonomous robots," *Robotics Autonomous Syst.* **82**, 1–14 (2016).
16. H. A. ElMaraghy and J. M. Rondeau, "Automated planning and programming environments for robots," *Robotica* **10**(1), 75–82 (1992).
17. M. Wang and T. Wu, "Cooperative co-evolution based distributed path planning of multiple mobile robots," *J. Zhejiang University Science* **6A**(7), 697–706 (2005).
18. Y. Dongyong, C. Jinyin, N. Matsumoto and Y. Yamane, "Multi-robot Path Planning Based on Cooperative Co-evolution and Adaptive CGA," *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, Hong Kong (2006) pp. 547–550.
19. R. Kala, "Multi-Robot Path Planning using Co-Evolutionary Genetic Programming," *Expert Syst. Appl.* **39**(3), 3817–3831 (2012).
20. X. Zhang, X. Guan, I. Hwang and K. Cai, "A hybrid distributed-centralized conflict resolution approach for multi-aircraft based on cooperative co-evolutionary," *Science China Information Sci.* **56**(12), 1–16 (2013).
21. D. Berenson, J. Kuffner and H. Choset, "An Optimization Approach to Planning for Mobile Manipulation," *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, Pasadena, CA, USA (2008) pp. 1187–1192.
22. P. Curkovic and B. Jerbic, "Dual-arm robot motion planning based on cooperative coevolution," *Emerging Trends in Technological Innovation, IFIP Advances in Information and Communication Technology*, vol. 314 (Springer, Berlin, Heidelberg, 2010).
23. E. K. Xidias and P. N. Azariadis, "Mission design for a group of autonomous guided vehicles," *Robotics Autonomous Syst.* **59**(1), 34–43 (2011).
24. E. Xidias, P. Zacharia and A. Nearchou, "Path planning and scheduling for a fleet of autonomous vehicles," *Robotica* **34**(10), 2257–2273 (2016).
25. R. Kala, "Coordination in navigation of multiple mobile robots," *Cybernetics Syst.* **45**(1), 1–24 (2014).

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press

26. J. Xiao, Z. Michalewicz, L. Zhang and K. Trojanowski, "Adaptive evolutionary planner/navigator for mobile robots," *IEEE Trans. Evolutionary Comput.* **1**(1), 18–28 (1997).
27. A. R. Shirazi and Y. Jin, "A strategy for self-organized coordinated motion of a swarm of minimalist robots," *IEEE Trans. Emerging Topics Comput. Intell.* **1**(5), 326–338 (2017).
28. R. Kala, "Sampling based Mission Planning for Multiple Robots," *Proceedings of the IEEE Congress on Evolutionary Computation, Vancouver*, BC, IEEE (2016) pp. 662–669.
29. M. Cefalo and G. Oriolo, "A general framework for task-constrained motion planning with moving obstacles," *Robotica* **37**(3), 575–598 (2018). doi: 10.1017/S0263574718001182.
30. I. Waheed and R. Fotouhi, "Trajectory and temporal planning of a wheeled mobile robot on an uneven surface," *Robotica* **27**(4), 481–498 (2009).
31. N. Tsiogkas and D. M. Lane, "An evolutionary algorithm for online, resource-constrained, multivehicle sensing mission planning," *IEEE Robotics Automation Lett.* **3**(2), 1199–1206 (2018).
32. G. B. Lamont, J. N. Slear and K. Melendez, "UAV Swarm Mission Planning and Routing using Multi-Objective Evolutionary Algorithms," *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making*, Honolulu, HI (2007) pp. 10–20.
33. L. Geng, Y. F. Zhang, J. J. Wang, J. Y. H. Fuh and S. H. Teo, "Mission planning of autonomous UAVs for urban surveillance with evolutionary algorithms," *Proceedings of the 2013 10th IEEE International Conference on Control and Automation (ICCA)*, Hangzhou (2013) pp. 828–833.
34. L. Yu and Z. Cai, "Robot Exploration Mission Planning Based on Heterogeneous Interactive Cultural Hybrid Algorithm," *Proceedings of the 2009 Fifth International Conference on Natural Computation*, Tianjin (2009) pp. 583–587.
35. H. Baalbaki, X. Xie and X. Delorme, "Mission assignment and scheduling for a team of service robots using evolutionary algorithms," *Proceedings of the 2010 IEEE Workshop on Health Care Management*, Venice (2010) pp. 1–6.
36. J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," *Proceedings of the Second International Conference on Genetic algorithms and their Application*, Cambridge, MA (1987) pp. 14–21.
37. J. E. Baker, "Adaptive selection methods for genetic algorithms," *Proceedings of the International Conference on Genetic Algorithms and their Applications*, Pittsburg, PA (1985) pp. 101–111.
38. J. J. Grefenstette and J. E. Baker, "How genetic algorithms work: A critical look at implicit parallelism," *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San Mateo, CA (1989) pp. 20–27.
39. W. Spears and K. de Jong, "On the virtues of parameterized uniform crossover," *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufman, San Mateo, CA, (1991) pp. 230–236,
40. L. E. Kavraki, P. Svestka, J. C. Latombe and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robotics Automation* **12**(4), 566–580 (1996).
41. R. Kala, "Homotopic roadmap generation for robot motion planning," *J. Intell. Robotic Syst.* **82**(3–4), 555–575 (2016).
42. R. Kala, Video Results for the paper Robot Mission Planning using Co-evolutionary Optimization (2019). Available at: https://youtu.be/DHa-22PjEVA.
43. R. Kala, Video Results for the paper Robot Mission Planning using Co-evolutionary Optimization (2019). Available at: https://youtu.be/931JVTud_Mg.
44. A. Cimatti, E. M. Clarke, F. Giunchiglia and M. Roveri, "NUSMV: A New Symbolic Model Verifier," Proceeding of the 11th International Conference on Computer Aided Verification (Springer-Verlag, London, UK, 1999) pp. 495–499.

https://doi.org/10.1017/S026357471900081X Published online by Cambridge University Press