Intelligent Service Robotics (2021) 14:741–771
https://doi.org/10.1007/s11370-021-00389-0

ORIGINAL RESEARCH PAPER

Multi-robot mission planning using evolutionary computation with incremental task addition

Rahul Kala[1]

Received: 22 November 2020 / Accepted: 7 October 2021 / Published online: 18 November 2021

Abstract
Mission planning asks the robot to solve complex missions, requiring the robot to execute several actions such that a complex goal condition (called mission) is satisfied. The missions can be specified by using Linear Temporal Logic or any custom domain-specific language. Here, we consider service robots in the home or office environments with multiple users, each specifying a task that together makes a mission. Conventional mission planning techniques assume that the complete mission specification is known in advance, while mission planning is incremental wherein tasks keep getting added and deleted. The exponential computational complexity of the current model verification-based approaches is not a viable solution. The paper presents an evolutionary framework for incrementally solving the problem of mission planning for the problems for which a polynomial-time verification system exists for the tasks. The optimal solution is represented in an encoding-specific format that is compiled to form a linear trajectory of the robots. Optimization is done as the robot operates. Thus, the robot at any time has a partial solution to every task that has already been executed that (along with the robot assigned to the task) cannot be altered, while a part representing the next steps of the robot to be optimized. This acts as a constraint in the continuous optimization. The proposed approach uses Dynamic Programming to integrate the solutions of tasks and as a distance function in the diversity preserving evolutionary computation since the mission constantly changes. Comparisons are made with a baseline evolutionary computation without Dynamic Programming, a non-incremental version of the proposed algorithm, and several greedy strategies. The proposed algorithm is seen to perform better than all other methods. Further, the algorithm is implemented and demonstrated on a Pioneer LX robot.

Keywords Mission planning · Robot motion planning · Linear temporal logic · Evolutionary computation · Diversity preservation

1 Introduction

Robots have been used for a very long time to do a variety of tasks like getting an object of interest from a site, inspecting a remote site, etc. A requirement from the modern-day robots is that the user should be able to instruct the robots, asking the robots to execute complex tasks like getting a variety of items spread in the home or office environment, cooking complex recipes, etc. All these operations require the robots to perform multiple operations in the correct sequence. Such a problem is called the mission planning problem. The mission is specified in a formal language by the user that can be understood by the robots. In this paper, we envision a team of robots operating for facilities like home or office with several users who continuously give new instructions to the robots to carry out several tasks. The robotic team must plan and distribute the work among themselves, while optimally solving the entire mission including adapting the solution as new requirements are added or changed continuously by the users.

The classic task planning approaches can be used to specify several missions using atomic propositions. The Linear Temporal Logic (LTL) [1, 2] allows specifying missions where the different atomic propositions have temporal constraints like one proposition should remain true until another proposition gets true. The classic planning techniques including the use of formal languages typically have an exponential complexity and heuristics does a little to make the solutions scalable. This is an ideal situation for the use of evolutionary

✉ Rahul Kala
rkala001@gmail.com
http://rkala.in

1 Centre of Intelligent Robotics, Indian Institute of Information Technology, Allahabad, India

computation as a planning system, which ensures probabilistic completeness, meaning the probability of finding a solution if one exists tends to one as time tends to infinity; and probabilistic optimality, meaning that the solution tends to the optimal solution as time tends to infinity. The evolutionary computation is used to search for an optimal sequence that solves the mission. For this, it is important that a polynomial-time verification of the solution is possible and hence the solver can only be used for some constrained languages. This is another problem associated with the model verification-based solvers for which even verification is an exponential complexity solution. Many real-life applications are now being solved by using evolutionary computation. The problem is encoded using problem-specific heuristics such that the solution is always valid or can be checked for validity in a polynomial time. As an example, for the Travelling Salesman Problem, a permutation of all places is a valid solution; for classic robot motion planning, a traversal of the path is done in a linear time to check for collisions, etc.

It must be noted that the paper takes only a general inspiration from LTL to specify the mission using a formal language and using a computational method to solve the synthesis problem. The paper itself does not use LTL or any of its variants. The paper does not use the LTL operators for the mission specification. LTL considers infinite traces, while the problem under consideration is for finite strings only. The paper allows the specification of a mission using the AND, OR, and sequencing operations only. A mission specified using these operations is polynomial verifiable and considers only finite strings. The mission planning problem is then the generation of a finite length string that is verified by the verification system, and preferably the string has the smallest path cost. Mission specified using only AND, OR and sequencing operations may appear to have very little expressive power. However, the paper only considers service robots that are useful to do the everyday chores of humans, and a lot of complex operations that humans may require to be done by the robots for everyday life are covered by these 3 operators. The language has not been checked for utility for other domains.

In this paper, the robot mission planning problem is being solved. A *task* is defined as an instruction given by a user consisting of Boolean and temporal constraints which must be solved by the robot. Numerous tasks coming from the same or different users constitute a *mission*. The use case is that there are numerous users in an office building. There is a fleet of robots catering to the needs of all the users. A robot solving any task sub-optimally may mean taking in hours for a task that could be otherwise completed in a very few minutes.

The incremental nature of the problem states that the users keep adding task specifications, as they may have requirements. A user may at any instance of time need a specific task (or a collection of tasks) to be given to the robot, while the robot may already be doing some other tasks of the same or other users. The robot neither knows about a prospective task coming before a user gives the task to the robot nor does the robot have a knowledge of the nature and timing of the future tasks that may come. Upon addition of a new task, a complete re-planning may not be possible due to the large computation times. Incremental synthesis approaches save time by reusing the models; however, the approaches must ensure computational efficiency using problem-specific heuristics.

The incremental nature of the solution is that the solution iteratively improves. After a solution is computed, the robot or robot fleet executes the solution. If the solver is iterative or anytime in nature, the significant amount of time when the robot is physically doing an operation can be used by the solver to further optimize the rest of the solution. The assumption is that an optimal search has an exponential complexity even after the use of heuristics while the robot cannot be asked to wait indefinitely for a solution. Therefore, the robot needs to make use of an approximation algorithm. There can be greedy heuristics that can be used to compute a reasonably good solution in a small duration of time. However, the robot has an unknown time during which it travels that can be used to compute the approximation solution. The aim is hence to use as many greedy heuristics as possible (like the use of Dynamic Programming), while to also allow the robot to benefit from the unknown time that the robot takes to travel from one place to the other that can be used for improving the solution quality. Many anytime algorithms exist in the literature. Like the Anytime-A* algorithm can continuously improve the path along with time. However, the Genetic Algorithm is specifically chosen as a solver that allows easy integration of the heuristics like Dynamic Programming for the proposed work.

The paper uses Dynamic Programming to integrate the solutions of different tasks to make the solution of a mission, which happens continuously during the optimization process. Hence adding tasks means retaining the old computations, while optimally adding the solution of the new task. A more difficult problem, however, arises in re-wiring the solution when the robot partially solves a task. The evolutionary solver iteratively optimizes solutions of the task; however, once a task is partially solved by the robot, neither can the robot assigned to the task change, nor the partial part of the task solved by the robot be changed. It may initially appear that deleting the solved components is an easy option; however, partial solutions cannot be verified by the verification system. Further, the problem is that the path traversed by the robot is linear, while the evolutionary individual is a complex encoding that is compiled by Dynamic Programming to get a linear sequence, and hence, the evolutionary individual representation domain is not the same as the final solution domain. This requires back tracing the solution to the evolutionary individual domain.

Since the mission specification will be continuously altered, both when a new task is added as well as when a task is partially solved, the evolutionary algorithm is executed with diversity preservation. The addition or deletion of a task changes the mission specification, which changes the objective measure used in the evolution. It is now possible for a local optimum to become a global optimum and vice versa. Diversity preservation techniques ensure that a diverse set of individuals are always available at different places in the fitness landscape to deal with the problems of changing fitness landscape. Here as well typicality is that diversity preservation requires a distance function, while the evolutionary individual is of variable length and hence an off-the-shelf distance function cannot be used. The Dynamic Programming-based edit distance is hence used for diversity preservation.

This is one of the first approaches to design a complete solution to incrementally solve the mission planning problem with multiple robots to the best of the knowledge of the author, wherein the solution and individual representation make two different domains, and deleting the traced path is not possible due to verifiability. At the same time, the paper stresses that the model verification tools for mission planning problems cannot be practically used for massively large problems possible in the domain of service robotics due to the exponential computational complexity and lack of optimality.

The main contributions of the paper are:

1. An incremental approach to mission planning using evolutionary computation is proposed. The evolutionary technique continuously optimizes the plan as the robot acts based on the best plan available. So, there is a variable time till the robot is executing the current operation, till when the next operation should be finalized as per the optimality criterion. This is a new approach to solve the problem, while the literature uses exponential complexity time solutions that cannot adjust to the variable time available. Tasks can be added at any time, while the tasks are deleted as the robot moves. The optimizer adjusts to the changing tasks. The algorithm ensures probabilistic optimality.
2. The solver continuously optimizes the current best plan as the robot acts. Thus, at any time a part of the tasks has already been executed while the remaining part is scheduled to be executed. The model verification engine can check for the validity of the string representing the complete task's solution and not a part string representing the solution of the task yet to be executed. The evolutionary approach is thus made freeze-aware, wherein the already solved parts of the tasks are frozen and need to be fixed by the evolutionary approach. The problem is unconventional since the solved string could be scattered in different parts of different genetic individuals.
3. Diversity preservation is added. Since the tasks are continuously added, deleted, or altered; the problem specification and thus the fitness landscape continuously changes. Diversity preservation helps to escape local optima upon a change in the fitness landscape. The genetic individual has a variable-length encoding and thus Euclidian and other distance functions cannot be directly used for diversity preservation. A Dynamic Programming-based distance function is proposed.
4. The solution is extended to multiple robots where the solver simultaneously optimizes the robot assignment to different tasks and the trajectory for each participating robot.

2 Background

2.1 Genetic algorithms

Genetic Algorithm is an evolutionary algorithm widely used to solve optimization problems. The algorithm is inspired by the natural evolutionary process, wherein a population of individuals mate to produce new offspring for the new generation. The process follows Darwin's principle of the survival of the fittest wherein the fitter individuals are stronger and mate more times while the weaker individuals normally die without mating. The new generation is generally healthier than the parent population, while the evolution process shows an adaptation of the individuals in response to the changing environment.

The Genetic Algorithm models an individual as a solution to the problem in an encoded format like a linear string called the genotype. The population is a collection of such individuals. The goodness of an individual is tested by using a problem-specific objective function called the fitness function. The algorithm iteratively improves the fitness value of the individuals, and every iteration of the algorithm is called a generation. To create the population pool of the next generation, the algorithm first selects a proportion of fitter individuals using a process called selection. Selection stochastically selects individuals such that the probability of selection of fitter individuals is higher that may get selected multiple times, while the probability of selection of the weaker individuals is lower that may not get selected. The selected individuals mate and exchange characteristics to produce children by an operation known as crossover. In a real-coded genetic algorithm, the individuals are encoded as a real numbered string. The arithmetic crossover in such a case produces children as a weighted addition of the parents. The individuals after crossover also undergo a mutation operation signifying the errors incurred in the generation of the children. For a real coded genetic algorithm, a Gaussian mutation can be used that adds a Gaussian noise to the real-coded genes. The parents and newly produced children compete, and the fittest individuals are selected to make the new population pool.

The initial population of the Genetic Algorithm may be generated by any greedy method signifying potentially good candidates. The initial population may as well be random. The individuals are initially scattered in the entire fitness landscape. The individuals converge towards the global optima with time because of the genetic operators. The algorithm attempts to maintain a tradeoff between exploration, which tries to expand the horizon of search of the individuals delaying convergence; and exploitation, which aims to quickly converge the individuals into the current optima.

Consider the problem of finding the shortest common super-sequence of two strings s_1 and s_2. Let $s_1(i)$ denote the substring till the ith character and similarly let $s_2(j)$ denote the substring till the jth character. Let the length of the shortest common super-sequence of $s_1(i)$ and $s_2(j)$ be $l(i,j)$. If the character $s_1(i)$ and $s_2(j)$ are the same, then we can use the solution of $l(i-1,j-1)$ or the shortest common super-sequence of $s_1(i-1)$ and $s_2(j-1)$ and add the character $s_1(i)$ making the solution of $l(i,j)$ as $l(i-1,j-1)+1$. If the character $s_1(i)$ and $s_2(j)$ are different, then we can either take the solution of $l(i-1,j)$ and add the character $s_1(i)$, or take the solution of $l(i,j-1)$ and add the character $s_2(j)$, whichever is smaller. The smallest case is when either of the strings is empty (size 0) and thus the solution is the other string. The solution is thus given by Eq. (1)

$$l(i,j)=\begin{cases} i & \text{if } j=0 \text{ (case 1 base)}\\ j & \text{if } i=0 \text{ (case 2 base)}\\ l(i-1,j-1)+1 & \text{if } s_1(i)=s_2(j)\wedge i>0\wedge j>0 \text{ (case 3)}\\ l(i-1,j) & \text{if } s_1(i)\neq s_2(j)\wedge l(i-1,j)\le l(i,j-1)\wedge i>0\wedge j>0 \text{ (case 1)}\\ l(i,j-1) & \text{if } s_1(i)\neq s_2(j)\wedge l(i-1,j)> l(i,j-1)\wedge i>0\wedge j>0 \text{ (case 2)} \end{cases} \tag{1}$$

Instead of solving the equation recursively, Dynamic Programming uses memorization to store all solutions of $l(i,j)$ so that every unique (i,j) pair is only solved once. Equation (1) gives the length of the super-sequence. To trace the super-sequence, additional information is stored in a similar data structure called parent (say π) that stores which case was used to compute the super-sequence, given by Eq. (2).

$$\pi(i,j)=\begin{cases} 1 \text{ if } j=0 \text{ (case 1 base)}\\ 2 \text{ if } \ =0 \text{ (case 2 base)}\\ 3 \text{ if } s_1(i)=s_2(j)\wedge i>0\wedge j>0 \text{ (case 3)}\\ 1 \text{ if } s_1(i)\neq s_2(j)\wedge l(i-1,j)\le l(i,j-1)\wedge i>0\wedge j>0 \text{ (case 1)}\\ 2 \text{ if } s_1(i)\neq s_2(j)\wedge l(i-1,j)> l(i,j-1)\wedge i>0\wedge j>0 \text{ (case 2)} \end{cases} \tag{2}$$

2.2 Dynamic programming

It is common to recursively divide a problem into smaller parts and to integrate the solutions of the smaller problems to make a solution to the larger problem. It is assumed that the solution of all unit problems is available (or can be computed in a constant time). Such divisions however can result in an exponential number of problems that may take an extremely long time to solve. Dynamic Programming uses a concept called memorization to store the solutions of all the sub-problems as they are solved. Thereafter, if a sub-problem gets generated again, its memorized solution is used directly, rather than computing the entire solution again. In this manner, Dynamic Programming significantly reduces the number of sub-problems by ensuring all problems are solved once.

3 Related works

A lot of work has been done using model verification tools for robot motion planning using LTL. In a pioneering work, Kress-Gazit et al. [3] used triangulation to decompose the map into a transition system which was fed into a Linear Temporal Logic solver for a robot to generate a controller for discrete and continuous domains. In the same vein, Lahijanian et al. [4] added the notion of probabilities to guarantee that the solution returned satisfies the minimum specified probabilities. The actual probabilities were learned by using Reinforcement Learning. Another related work by Bhatia et al. [5, 6] presented the same problem as an integration of a low-level controller and a high-level mission planner, wherein the low-level controller used a sampling-based approach incorporating robot dynamics. Heuristics can be used to get a faster solution, which was shown in the work of McMahon and Plaku [7]. The heuristics used are aimed at getting solutions early as well as at low cost. In the same vein, Svorenova et al. [8] incorporated preferences and rewards and intensified the search towards the goal as well as maximizing rewards. These approaches represent the LTL in the form of an automaton, which has an exponential complexity thus limiting the use to only smaller problem sizes. The approaches further use a product of the transition system with the automaton representation to ensure that only valid transitions are represented which is again an exponential complexity operation. The integration with low-level planners and incorporation of uncertainties further complicates the search over such representations. The aim of the paper is thus to use a restricted language that can be verified in a polynomial time. The search for an optimal solution will still have an exponential complexity and thus the evolutionary algorithm is used to make an iterative algorithm that continuously optimizes the solution. It must be noted that the proposed approach does not solve for the full LTL and only the AND, OR and sequence operators can be used in the proposed language. The proposed approach argues that given the existing approaches on full LTL cannot deal with exceptionally large problem sizes, research efforts are required to custom design languages even if with restricted representational ability that can be verified in a polynomial time. Uncertainty cannot be currently handled in the proposed approach since it is not possible to compute the uncertainty metrics within a polynomial-time of the number of variables.

Some recent approaches are optimality conscious [9, 10]. The approaches present the problem as a LTL that is converted into an automaton. To model only the feasible transitions, a product of the automaton is taken with the transition system. Here, the conversion of the temporal logic into the automaton is exponential in time and space with respect to the variables used, and therefore, the product with the transition system is also exponential. This however converts the problem into a graph search problem over the product automaton. In the work of Svorenova et al. [11], the notion of probabilities is taken in addition, over which an exhaustive search finds the optimal solution while considering the probabilities. In another recent work, Ulusoy et al. [12] used product automaton to plan for multiple robots optimally. The authors further synchronized the motion of the robots and corrected the plans when the robots were out of synchronization. Fu et al. [13] added heuristics while searching for the optimal plan. The heuristics resulted in focussing on the paths which are more likely to get to the terminal stage. The approaches use the LTL language to take the input while the user can give any formula specified using the LTL. The papers have different system modelling and different optimization objective that is suited to the specifics of the application. Since the input is a generic formula, its verification and hence the search is over a space that has exponential complexity. Further, the approaches need the computation of the product automaton that also has exponential complexity. The proposed work on the contrary attacks problems where an exponential complexity is not feasible due to many proposition variables. The proposed approach restricts the language to be polynomial verifiable and thus facilitating a solution using evolutionary computation. Only AND, OR, and sequence operations can be used, the use of which makes a language that is polynomial verifiable. Limiting the expressive power is important to avoid exponential complexities.

Lacerda et al. [14] considered co-safe LTL, where the language has only those operators that generate a good prefix that appended to an infinite length action sequence guarantee the satisfiability of the formula. Practically, the language reduces the problem of search from infinite sequences to only finding the finite good prefix. The authors specifically allowed tasks to be added to the mission while the tasks were deleted on completion. The system planned dynamically and adjusted to the changes. Schillinger et al. [15] assumed that the mission consists of several tasks and jointly performed the robot allocation and task planning like the proposed approach. The authors used co-safe LTL while working over the computation of the finite prefix to generate optimal trajectories as per the cost function. The authors handled constraints like resources in the LTL formulation. Faruq et al. [16] further incorporated uncertainties in their modelling, while trying to compute a policy that maximizes the probability of completion of the mission. To make the approach computationally efficient, the problem was solved through local Markov Decision Processes for the individual robots with a switching mechanism for task allocations. The approaches still suffer from the generic problems associated with the LTL of a high computational time and exponential complexity that is worsened because of the uncertainties. The approaches are not scalable to many mission sites and robots. Thus, the paper approaches the problem using an optimization approach while making the language specific to be polynomial verifiable. Even though uncertainty makes the solution realistic, it is currently not possible to compute the same in a time linear to the number of variables and is thus omitted. For the same reasons only AND, OR, and sequence operators are used.

Torres and Baier [17] enabled the translation of an LTL expression into an alternate representation in a polynomial time that also facilitated the addition of new constraints with time using the primitives of alternating automata. This was done by using additional fluents. However, the translation does not ascertain the representation of all possible transitions that can severely affect the optimality, even though getting a feasible solution may be possible. Camacho et al.

[18] also aimed to represent a general LTL formula as a PDDL, while the conversion could theoretically be exponential. The conversion still requires a search for an optimal solution without good heuristics that can be exponential. Menghi and Garcia [19] considered sub-missions to be solved by the different robots in a decentralized manner in a partially known environment. The robots could plan to meet, which was also associated with uncertainty. The authors' performed optimistic and pessimistic planning of every robot, handling the meeting constraints. Decentralization and decomposition enable reducing the computation time. However, a classic search over the problem itself could be exponential, especially when done with an aim of computing plans with the optimal cost.

Sampling is a natural response to deal with systems with a massive complexity. Sampling has also been applied to mission planning. A good approach [20] is to sample out trees in the transition system, which is used to synthesize a feasible and optimal path that solves the mission. A random configuration is used to grow the tree, similar to the Rapidly-exploring Random Tree [21, 22] formulation. Only feasible sequences are expanded. In another similar work Kantaros and Zavlanos [23, 24] proposed strategies to generate the samples and grow the trees while searching for a solution that satisfies the temporal logic specification. Sampling can significantly prune the search and result in a much faster algorithm. However, the verification still requires a conversion of the temporal logic into an automaton, which can be exponential in terms of the number of variables used to represent the system. Here, the focus is on the problem when the conversion of the mission formula into an automaton is itself infeasible in the first place. The proposed work restricts the language to the use of AND, OR, and sequence only that is polynomial verifiable, while the restriction is needed as otherwise the problem with an exponential complexity cannot be solved optimally for very large problem sizes.

The problem of mission planning has particularly not been solved by Evolutionary Computation. A particular work to discuss includes Xidias and Azariadis [25] wherein the mission was to visit several sites by multiple robots using a centralized Genetic Algorithm. Lu and Yue [26] solved the multi-robot Travelling Salesman Problem by using a decentralized technique of allocating sites, which was optimized by using Ant Colony Optimization. In the same problem, the notion of preference was added by Kala [27] while solving by using decentralized evolutionary computation. Senol [28] accounted for the human–robot interactions while optimizing a mission using a mixed-integer programming technique, without Boolean and temporal constraints. The problem with all the current approaches is that the temporal operators are rarely modelled and accounted for, which restricts the generality of the solver and thus the solver can be used in a few limited scenarios only. Using a single robot, a prior work of the author [29] incorporated the specification of Boolean and temporal constraints in the robot. The approach inculcated a sequencing temporal constraint into a generic Travelling Salesman Problem. The paper used no heuristic like Dynamic Programming that is included in the proposed work. The paper did not have any incremental nature and could not incorporate multiple tasks. The paper forms a very initial work of the author in the domain.

It is also worth noting the approaches used by the classic planning techniques, which form another school of learning to approach the same problem. The classic planning approaches are relevant to the work because many mission problems can also be specified as classic planning problems. Traditionally costs were not incorporated in the classic planning techniques and optimality was not considered in these approaches. Recently, especially with the increased use of PDDL3 [30, 31], the costs are now being actively taken into consideration. The problem with the approaches is again the absence of the temporal operators. Handling temporal constraints in classic planning techniques is quite different from the ones in temporal logic. A naïve user can write an LTL formula denoting the mission; however, conversion of the same formula into a format that can be directly fed into a classic planning library is still not possible for a naïve user. This makes classic planning not preferable for robot mission planning, where the mission needs to come from an end-user. The proposed approach considers the costs of travelling from one place to the place where the action needs to be performed, and additionally the cost of the action. The domain also faces the problem that the actions are continuous and geometric for the robot, however, are abstracted to Boolean states in classic planning. The integration of tasks and geometric planning [32–34] is another good field of study that has recently gained momentum. The motion obtained by any geometric planning algorithm can be used by any low-level reactive planning or control algorithm for the motion of a team of robots. In the proposed approach, the cost between every pair of consecutive actions shall eventually be queried by the planner for feasibility and cost. On the other hand, the number of actions is small and finite. Therefore, a cost matrix is computed at the start instead of computing it on the fly during planning. Artificial Potential Field [35, 36] is a popular choice for moving robotic teams. A similar approach is also used in the proposed work.

Lagoudakis et al. [37] considered a mission where a team of robots had to visit all the mentioned sites. The authors implemented a multi-round auction, where the robots cast a bid using Prim's heuristic method. The method is dynamic, and changes are rectified in the future rounds of the auction. The specific problem has a good heuristic available in the form of a Prim's algorithm that gives an optimality guarantee, while the proposed approach has no such heuristic, and optimization is used instead. For the same reasons, instead of a heuristic assignment of robots to the tasks, the robot allocation is also done as a part of the same optimization problem.

4 Problem definition

The problem is that several users have given several instructions to a team of robots called tasks. The tasks are written in a formal language and specify the operations to be done by the robot with ordering and choices that affect the optimality. The robots must distribute the tasks among themselves optimally and plan the tasks optimally. The solver requires a verification system that checks if a string satisfies the constraints of the language. The problem is then to generate the optimal sequence as per the objective criterion that adheres to all the task constraints. The challenge in the problem is that the tasks shall be continuously added by different users, while an accomplished task is deleted from the mission specification. The solutions are thus continuously adapted as per the latest mission specification. The robots continuously work and accomplish different tasks, while the time of operation of the robotic team is used to further optimize the mission plans. The robotic team at any stage will have partially completed tasks, while the optimizer should not alter the part of the plan that has already been accomplished. The different aspects of the problem are given in the following sub-sections.

4.1 Overall problem

The problem is to solve a mission (χ) which is a collection of tasks (ψ), that is $\chi = \{\psi_1, \psi_2, \psi_3, \dots \psi_n\}$, where n is the number of tasks. It is assumed that the mission is dynamic and changes with time, including the addition of a task ψ by a user ($\chi \leftarrow \chi \cup \{\psi\}$) or completion of a task completely ($\chi \leftarrow \chi - \{\psi\}$). The task ($\psi$) consists of atomic operations with some Boolean and temporal constraints. An atomic operation is the unit operation that the robot can be asked to perform like to visit a place, pick and place an item, or operate a machine. The unit operations are non-divisible, meaning the robot cannot execute any other action till the current atomic operation is complete. A string s consists of an ordered list of atomic operations [σ_1, σ_2, σ_3, … $\sigma_{\text{len}(s)}$], where len(s) is the size of the string. The set of all atomic operations $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \dots \sigma_m\}$ is a part of the problem specification, where m is the number of possible atomic operations. It is assumed that a verification engine exists such that it can be ascertained whether string s satisfies the requirements of the task ψ in polynomial time, or $s \vDash \psi$ is polynomial computable. The approach is generic to the use of any Boolean and temporal operator, provided that the resultant language is polynomial verifiable. However, for this paper, we restrict the operators to sequencing (then or T), AND operator ($\wedge$), and OR operator ($\vee$). The task ψ can be recursively enumerated using these operators using the Backus Naur Form given by Eq. (3)

$$< \psi > := (< \psi >) | < \psi > T < \psi > | < \psi > \wedge < \psi > | < \psi > \vee < \psi > | \sigma_1 | \sigma_2 | \sigma_3 | \dots | \sigma_m \tag{3}$$

The semantics of the language that checks if the string s satisfies the task ψ ($s \vDash \psi$) is given by Eqs. (4–7)

$$s \vDash \psi_1 T \psi_2 \text{ iff } \exists_i s(1:i) \vDash \psi_1 \wedge s(i+1:\text{end}) \vDash \psi_2 \tag{4}$$

$$s \vDash \psi_1 \wedge \psi_2 \text{ iff } s \vDash \psi_1 \wedge s \vDash \psi_2 \tag{5}$$

$$s \vDash \psi_1 \vee \psi_2 \text{ iff } s \vDash \psi_1 \vee s \vDash \psi_2 \tag{6}$$

$$s \vDash \sigma_i \text{ iff } \sigma_i \in s \tag{7}$$

Equations (4–7) break down a task ψ recursively into smaller sub-tasks ψ_1 and ψ_2. Equation (4) states that a string satisfies ψ_1 then ψ_2 if a prefix till the ith character $s(1{:}i)$ satisfies ψ_1 and the remaining string after the ith character $s(i+1{:}\text{end})$ satisfies the task ψ_2, which coincides with the literal definition of then. Equation (5) states that a string s satisfies the task ψ_1 and ψ_2, if the string simultaneously satisfies ψ_1 as well as ψ_2. Equation (6) states that a string s satisfies the task ψ_1 or ψ_2, if it either satisfies ψ_1 or instead satisfies ψ_2, or both. Equation (7) states that a string satisfies the unit operation σ_i if the string somewhere at any place asks the robot to perform the operation σ_i.

A verification system is made that checks for the satisfiability of any general string for a formula specified using these operators. The verification system is used to check for the feasibility of the sequence, while the optimizer attempts to find the action sequence for the tasks, whose Dynamic Programming based fusion gives the trajectories of the robot used for cost computation.

It is assumed that there are several robots $R = \{r_1, r_2, r_3, \dots, r_z\}$ available that collectively solve a mission, where z is the number of robots. However, a task may require a robot to physically carry objects, which cannot be transmitted across robots on the fly. Therefore, the entire task ψ must be solved by any robot, but only one robot. Tasks that do not have such constraints can be supplied as multiple tasks. Let ξ: $\chi \rightarrow R$ be the function that maps a task (ψ) to a robot $r = \xi(\psi)$.

Many practical scenarios were considered before putting a restriction on not allowing multiple robots to partly solve a task. If a task can be divided into multiple robots, the system would accept it as multiple tasks rather than a single task. Consider that a person wants 5 items (say A, B, C, D, and E) from 5 different places. Now it is possible to say that this is 1 task that can be easily divided by the robots for the sake of efficiency. However, the system will interpret it as 5 tasks (get A and deliver to the user; get B and deliver to the user; get C and deliver to the user; get D and deliver to the user; and get E and deliver to the user). The optimizer will consider several possible task assignments among robots, including assigning some of the items to one robot and some others to another robot.

Let $T' = \{\tau'_1, \tau'_2, \tau'_3, \ldots \tau'_z\}$ be the order of performance of atomic operations by all robots where the subscript denotes the robot number (say r). $\tau'_r = \left[\sigma^r_0, \sigma^r_1, \sigma^r_2, \ldots \sigma^r_{\mathrm{len}(\tau'_r)}\right]$ specifies the order of atomic operations to be performed by the robot r, where len(τ'_r) is the number of operations. The $'$ is intentionally added to representation to annotate that the trajectory is at the symbolic level and not the real continuous trajectory traced by the robot. Let c: $\Sigma^2 \to \mathbb{R}+$ be the cost function such that $c(\mathrm{loc}(\sigma_i),\sigma_j)$ is the cost in physically going from the mission site of σ_i (called loc(σ_i)) to the mission site of σ_j and performing the operation σ_j. Let S_r be the current location of robot r. $c(S_r,\sigma_j)$ denotes the cost of going from the current location of robot r to the mission site of σ_j and performing the operation σ_j. Here, the cost function is taken as the time for modelling. The total cost $C(r)$ incurred by the robot r is given by Eq. (8)

$$C\left(\tau'_r\right) = c\left(S_r, \sigma^r_1\right) + \sum_{i=1}^{\mathrm{len}(\tau'(r))-1} c\left(\mathrm{loc}\left(\sigma^r_i\right), \sigma^r_{i+1}\right) \quad (8)$$

Here len() denotes the length of the solution. Since the cost is taken as the time, the aim is to solve all tasks in as little time as possible. The optimization objective is the makespan, or the total duration of time till the last working robot completes the last operation completely. The total cost of all robots combined is hence given by Eq. (9).

$$\mathrm{cost}\left(\mathrm{T}'\right) = \max_{r\in R}\left(C\left(\tau'_r\right)\right) \quad (9)$$

Let $\tau_{\mathrm{loc}(\sigma i),\mathrm{loc}(\sigma j)}$: $[0,1] \to \varsigma^{\mathrm{free}}$ be the trajectory from the mission site of σ_i (loc(σ_i) or source) to the mission site of σ_j (loc(σ_j)). Here, $\varsigma^{\mathrm{free}}$ is the free configuration space of the robot, or the set of configurations such that the robot neither collides with the obstacles or itself. In our case, the robot's configuration is defined by its position and orientation or (x,y,θ). The configuration space is all possible configurations that the robot can take or the set of all (x,y,θ) possible, commonly referred to as SE(2) or the Special Euclidean Group of order 2. However, a subset of these configurations will involve a collision between the robot and some obstacles detected by collision checking libraries, called the obstacle-prone configuration space or ς^{obs}. The remaining configurations constitute the free configuration space or $\varsigma^{\mathrm{free}}$ where the robot does not collide with any obstacle. $\varsigma^{\mathrm{free}}$ is the set complement of ς^{obs} over the super-set ς or $\varsigma^{\mathrm{free}} = \varsigma \backslash \varsigma^{\mathrm{obs}}$. Computing the trajectory is a classic robot motion planning problem [38, 39], wherein the problem is to compute a trajectory from the origin $\tau_{\mathrm{loc}(\sigma i),\mathrm{loc}(\sigma j)}(0) = \mathrm{loc}(\sigma_i)$ to the destination $\tau_{\mathrm{loc}(\sigma i),\mathrm{loc}(\sigma j)}(1) = \mathrm{loc}(\sigma_j)$, such that all intermediate points are collision-free $\tau_{\mathrm{loc}(\sigma i),\mathrm{loc}(\sigma j)}(e) \in \varsigma^{\mathrm{free}}$, $0 \le e \le 1$. The cost function is taken as the time required to trace the trajectory and operating, or $c(\mathrm{loc}(\sigma_i),\sigma_j) = \|\tau_{\mathrm{loc}(\sigma i),\mathrm{loc}(\sigma j)}\|/v + op(\sigma_j)$, where $\|\tau_{\mathrm{loc}(\sigma i),\mathrm{loc}(\sigma j)}\|$ is the total distance of the trajectory, v is the average speed of the robot and op(σ_j) is the cost of performing operation σ_j. It may be noted that this trajectory and costs are computed assuming a static map only and the actual time may be larger if there are other humans and dynamic obstacles. The trajectories and costs for all mission sites can be computed in advance with the static map of the place.

The problem is hence to find the optimal sequence for every robot and the optimal robot assignment to every task, or Eq. (10)

$$T^*, \xi^* = \arg\min_{\mathrm{T}',\xi} \mathrm{cost}\left(\mathrm{T}'\right) \quad (10)$$

such that all tasks in the mission are satisfied, or Eq. (11)

$$\tau'_r \vDash \psi, \quad \forall r \in R,\ \tau'_r \in \mathrm{T}',\ \psi \in \chi : \xi(\psi) = r \quad (11)$$

4.2 Incremental nature

Since the missions are continuously added by the user and deleted, the evolutionary optimization must run in parallel. Adding a new task by the user ($\chi \leftarrow \chi \cup \{\psi\}$) and deleting the old task completed by the robot ($\chi \leftarrow \chi - \{\psi\}$) are not difficult and do not change the problem formulation. However, suppose the robot trajectory τ'_r solves the task ψ, where $r = \xi(\psi)$. The robot starts the execution of the trajectory. Suppose at any time t, $\tau'_r(1{:}\varphi_r(t))$ part of the trajectory is already executed. Here, φ_r: $\mathbb{R} \to [0,\mathrm{len}(\tau'_r)]$ is the function that maps the time to the index of the planned trajectory at the symbol level. So, at time t, the robot will be executing the operation $\tau'_r(\varphi_r(t))$. The mapping can be understood by the ideal case wherein the robot operates at the assumed average speed of v. Hence, at time t, the robot must be executing the operation given by Eq. (12).

$$\varphi_r(t) = \begin{cases} i+1 : \left(c\left(S_r, \sigma^r_1\right) + \sum_{j=1}^{i} c\left(\mathrm{loc}\left(\sigma^r_j\right), \sigma^r_{j+1}\right) > t\right) & c\left(S_r, \sigma^r_1\right) \ge t \\ 1 & c\left(S_r, \sigma^r_1\right) < t \end{cases} \quad (12)$$

The iterator j iterates over the complete trajectory of robot r that is present as a sequence of operations $\left[\sigma^r_1, \sigma^r_2, \ldots \sigma^r_{\mathrm{len}(\tau'_r)}\right]$ with the source of the robot as S_r. The cost of the trajectory till the i + 1th item of the sequence is thus $c\left(S_r, \sigma_1^r\right) + \sum_{j=1}^{i} c\left(\sigma_j^r, \sigma_{j+1}^r\right)$. If for any operation this cost is more than the current time t, it means that the i + 1th operation for the robot is being executed, which is the output or the value of $\varphi_r(t)$.

It is worth noting that the index updates when the robot starts executing an operation and not when it completes executing the operation. The trajectory $\tau'_r(1{:}\varphi_r(t))$ has already been executed and therefore cannot be altered. This part of the trajectory is said to be *frozen* for the optimization process. Furthermore, unless the task execution has not yet started, or $\tau'_r(t_0{:}t) = \emptyset$, the robot assignment $r = \xi(\psi)$ cannot change, which is an added constraint for the solver. It must be stressed that Eq. (12) assumes that the actual time taken by the robot to complete every operation is known. However, the cost function only gives an estimate, and the actual robot may take a longer or a shorter time. As an example, the estimate to take the signature from a person maybe 1 min, while it may take longer as the person is talking over a phone. Hence, Eq. (12) is not used to estimate the index of the operation in the trajectory that is currently being executed. The index is calculated by making the robot signal the successful completion of a task.

The mechanism of freezing is such that the operations are non-preemptive. That is once a robot starts executing an operation and later a sequence is found such that it is better to leave the current operation unfinished and start with the new operation, the robot shall not leave the operation. This is because the user may have been notified for the start of the operation and cancellation may not be good for acceptance. Further, for many robots, such termination of operations may not be feasible. A preemptive version can also be implemented by a small change of code.

4.3 Solving by decomposition

If optimization is carried directly onto the problem, it shall become a centralized evolutionary computation that is known not to work unless the number of robots, operations, and tasks constituting the mission is very small in number. The absence of good heuristics like clustering for a Travelling Salesman Problem, further makes it hard to decompose the mission into simpler sub-missions to be solved by using a decentralized technique. Using centralized evolutionary computation further means that the addition of a mission does not mean that the past genetic individual can be easily modified to accommodate the new mission, nor does deletion of a mission mean that the relevant variables can be deleted from the corresponding individuals since they may be widely distributed in the entire individual that cannot be traced.

To best capture the heuristics of the mission, that a mission constitutes of multiple independent tasks, and to aid in the incremental addition and deletion of tasks in the mission, the problem is decomposed into two components that work hand in hand, one is to solve independently for all tasks and the second is to integrate the solutions of the tasks to make a solution of the mission.

In the reformulated problem, let τ_ψ^{task} be the sequence of operations to be performed by the robot $r = \xi(\psi)$ for the task ψ, such that $\tau_\psi^{\text{task}} \vDash \psi$. The path to be taken to solve the task is hence given by $\tau_\psi^{\text{task}} = \left[\sigma_0^\psi, \sigma_1^\psi, \sigma_2^\psi, \ldots \sigma_{\text{len}\left(\tau_\psi^{\text{task}}\right)}^\psi\right]$, where $\text{len}\left(\tau_\psi^{\text{task}}\right)$ is the length of the task solution. Let the sequence of operations to be performed by the robot be given by $\tau'_r = \left[\sigma_0^r, \sigma_1^r, \sigma_2^r, \ldots \sigma_{\text{len}(\tau'_r)}^r\right]$. The only constraint on τ'_r is that it must satisfy all tasks, that is Eq. (13)

$$\tau'_r \vDash \psi \forall \psi \in \chi : \xi(\psi) = r \tag{13}$$

Many realistic applications including the operation of robots in home and office environments have the property that tasks are given to the robot, instead of asking the robot not to do things, meaning that the language is negation-free. The primitive negations like avoiding obstacles, not moving to a hazardous region, etc. can be handled by the path planner. For such languages, if a string satisfies a task, any super-sequence of that string also satisfies the task. More formally, consider a general string s which is a subsequence of a string P. If s satisfies the task ψ, then P also satisfies the task ψ. Let $s|P$ denote s as a subsequence of P, or Eq. (14) holds

$$s \vDash \psi \Rightarrow P \vDash \psi, \quad \text{if } s|P \tag{14}$$

This is formally given in the "Appendix" section as Theorems 1–5. If a robot r has a set of tasks assigned to it, the robot trajectory (say, τ'_r) that is a super-sequence of all task trajectories $\left(\tau_\psi^{\text{task}}\right)$ satisfies all tasks given to the robot as per theorem 5. In other words, Eq. (15)

$$\tau_\psi^{\text{task}} \vDash \psi \Rightarrow \tau'_r \vDash \psi, \quad \text{if } \tau_\psi^{\text{task}} | \tau'_r \quad \forall \psi \in \chi : \xi(\psi) = r \tag{15}$$

The equation should be read as if the string τ_ψ^{task} satisfies the task ψ, then a superstring (say $\tau_r^{'}$) of τ_ψ^{task} (denoted by $\tau_\psi^{\text{task}} | \tau'_r$) also satisfies the task ψ, for all such tasks ψ assigned to the robot r. It is given that all task trajectories satisfy the given task, or $\tau_\psi^{\text{task}} \vDash \psi$. We consider only the sub-set of tasks assigned to the robot r, or the tasks for which $\xi(\psi) = r$. For all such tasks, since $\tau_{\psi}^{\text{task}} \vDash \psi$, a super-sequence of $\tau_{\psi}^{\text{task}}$, say $\tau_r^{'}$ $\left(\tau_{\psi}^{\text{task}} | \tau_r^{'}\right)$ also satisfies ψ $\left(\tau_r^{'} \vDash \psi\right)$ as per theorem 5. Further, we prove that the addition of new terms to $\tau_r^{'}$ apart from those necessitated as per the definition of super-sequence are unnecessary by theorem 6 given in the "Appendix" section.

The implied constraint on the construction of the trajectory of the robot is hence to construct a super trajectory of all trajectories of the tasks, such that no extra element is added. This trajectory and constraints are put into Eqs. (10) and (11) to get the optimal path. In other words, instead of directly computing the robot sequences τ'_r, the optimizer calculates the task sequences $\tau_{\psi}^{\text{task}}$ at one place and simultaneously fuses task sequences $\tau_{\psi}^{\text{task}}$ to produce the robot sequences τ'_r, at the other place. This means the addition of a task $(\chi \leftarrow \chi \cup \{\psi\})$ is simple since the optimizer will add a separate sequence for $\tau_{\psi}^{\text{task}}$. Similarly deleting a task $(\chi \leftarrow \chi - \{\psi\})$ is simple, in which case corresponding $\tau_{\psi}^{\text{task}}$ is simply deleted.

However, a larger problem is when the robot part solves a task, by executing trajectory $\tau'_r(1{:}\varphi_r(t))$. Now freeing $\tau'_r(1{:}\varphi_r(t))$ is not an option since the optimization happens on $\tau_{\psi}^{\text{task}}$. This requires a mechanism to link $\sigma \in \tau'_r$ to $\sigma \in \tau_{\psi}^{\text{task}}$.

The negation-free constraint (or the constraint that the robot can only be given operations to be performed that can be specified without using the negation) has an important implication. This makes the tasks non-conflicting. Therefore, achieving one task does not make the other task unsatisfiable. If a future task is added and correspondingly some operations are added to achieve the same task, the negation-free constraint suggests that the operations do not make any of the previous tasks unsatisfiable. The solver incrementally adapts to the tasks that come. If the tasks were not conflict-free, a future task could have made a previous task infeasible, requiring a solution to be re-computed. In such a case, the incremental approach would not effectively work, and the solution could have been as poor as a complete re-planning.

5 Solution design

5.1 Overall approach

The overall methodology is given in Fig. 1. The genesis of the solution is a mission-solving system that runs in the background and constantly adapts the current mission solution as per the altered mission specifications, addition of tasks by the user, and completion of (part-)tasks by the robots.

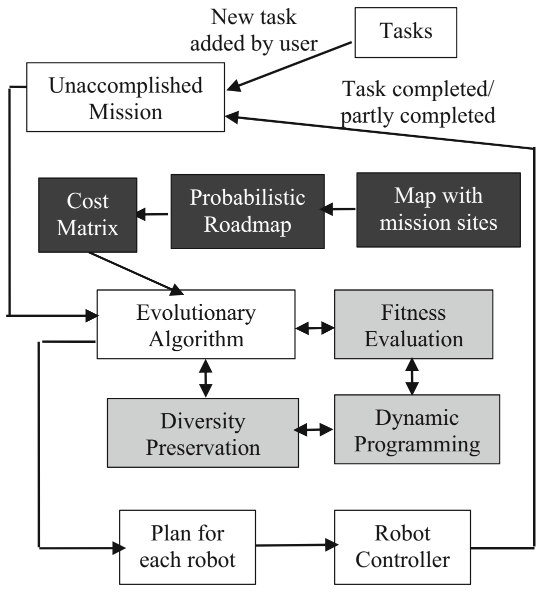


Fig. 1 Overall working methodology. Non shaded is the simplified programme flow

Since evolutionary computation is used, the solution keeps improving. This does not cause a computational overhead to the system since the optimization for the next operation happens while the robot is physically running and executing the current operation. Consider that the robot plans according to which it gets its next atomic operation to perform. Even before the robot starts executing the next atomic operation, the operation is marked as frozen, meaning the optimizer cannot change the next atomic operation. The robot shall take some time to physically reach the site where the operation needs to be done and to do the operation, sometimes alongside a human. The time required for the same is unknown. As the robot operates, the optimizer keeps optimizing the plan, keeping the frozen parts fixed. When the robot completes the current operation, it fetches the next operation as per the current best plan, freezes the operation, and executes the operation while still optimizing the rest of the plan. Since the mission specification will change often, the diversity preservation mechanisms are added to the evolutionary algorithm. Also, it may be possible that the mission has some heuristics specific to the domain of application, in which case the initial population generation needs to be done in a strategized manner to exploit any possible heuristic.

The tasks are added to the system by interrupts generated in the software. Upon receiving such an interrupt, the mission specification variables and all individuals in the population are altered to account for the new task added. Similarly, an interrupt is generated when a robot completes an operation in the task. In such a case, the variables of the mission specification and population pool are traced from the completed operation to freeze them from alteration. A special case is when the entire task completes, wherein again the mission specification variables and population pool is altered to delete the relevant variables.

The individual representation encodes for all tasks in the mission. The solution is needed as the trajectory of the robot. The fitness is itself calculated in terms of the trajectory of the robot. These are two different domains. Therefore, task-encoded solutions need to be compiled into the trajectory of the robot. This compilation is done optimally by using Dynamic Programming. Dynamic Programming only ensures an optimal fusion given some task trajectories; it does not ascertain optimal task trajectories in the first place, which is done by the evolutionary computation. Hence, Dynamic Programming is called within the fitness evaluation of the Evolutionary Computation. A similar Dynamic Programming is also used as the distance function of the diversity preservation, which is otherwise difficult since the different individuals will have different lengths representing the solutions of the tasks.

The output of the Evolutionary Algorithm is the plan or the sequence of steps to be taken by every robot. The algorithm sequentially gives the next operation of the plan to the robot for execution, while the rest of the plan is further optimized as the robot works. The execution of the operation (like going to a place and picking a book, taking a signature, etc.) is a capability that is assumed to be present with the robots and can be invoked as a library function, referred to as the robot controller.

The algorithm requires a cost function between every pair of possible operation site to compute the total cost of the plan that is optimized. The cost between every possible operation site pair is stored as a cost matrix. For the same, first, the map of the arena is made using SLAM techniques. Thereafter, in the static map, the Probabilistic Roadmap [40] technique is used to compute the cost matrix between every pair of mission site possible. The operational cost of physically operating is currently kept as 0.

5.2 Individual representation

The genetic individual encodes the solutions to the different tasks $\left(\tau_{\psi}^{\text{task}}\right)$, where a solution of each task consists of a string. The complete individual is given in Fig. 2. Some of the operations may have already been performed by the robot, and therefore, the associated variables need to be frozen which is also specified in the individual representation. The frozen components are non-optimizable. The individual also encodes the robot assignment for the tasks (ξ), which are also frozen for the tasks for which even a single step has been performed. The individual in such a genotypic form needs to be compiled into the trajectory of the individual robot in the phenotypic part which is done by using Dynamic Programming. However, a direct compilation would again make the algorithm have exponential complexity and hence the fusion is done by taking two tasks at a time in an ordered sequence. Let O: $[1,n] \rightarrow \chi$ be the function that maps every index (order) to a task, $O(i) = \psi$, which means that ψ will be the ith task to be presented to Dynamic Programming. This will be explained when dealing with Dynamic Programming. The ordering is also optimized in the evolutionary process and represented in the individual.

The genotype consists of a prospective solution as a sequence of operations for every task with the completed operations shown in grey, along with the order of fusion and robot assignment. Each robot needs a linear sequence of operations (phenotype), based on which the goodness of an individual is evaluated. The conversion of genotype to phenotype happens by noting the tasks assigned to every robot and sorting the tasks as per the order values. The solution trajectory for the task (from the genotype) is then fused by a Dynamic Programming technique in the sorted order of tasks. After fusion for all tasks, the fused trajectory becomes the phenotype that can be used for fitness evaluation.

5.3 Dynamic programming based fitness evaluation

Let us first discuss the fusion of the solutions of two tasks $\tau_{\psi_1}^{\text{task}}$ and $\tau_{\psi_2}^{\text{task}}$, to produce a superstring that satisfies both tasks. Let the Dynamic Programming fusion be denoted by $\tau_{\psi_1}^{\text{task}} \oplus_D \tau_{\psi_2}^{\text{task}}$. Let $d(i,j)$ denote the cost incurred in the fusion of $\tau_{\psi_1}^{\text{task}}$ till ith position and $\tau_{\psi_2}^{\text{task}}$ till jth position, that is the optimal fusion of $\tau_{\psi_1}^{\text{task}}(1 : i)$ and $\tau_{\psi_2}^{\text{task}}(1 : j)$. The calculation of $d(i,j)$ can be done by using Dynamic Programming; however, the application also needs the fused string. The string fused by Dynamic Programming is calculated by storing the sub-problem that leads to a solution of $d(i,j)$ in a new data structure called the *parent* or $\pi(i,j)$. More specifically, $\pi(i,j)$ is 1 if the last operation in the optimal calculation of $d(i,j)$ came from $\tau_{\psi_1}^{\text{task}}$ and 2 if the last operation instead came from $\tau_{\psi_2}^{\text{task}}$. For distance computation reasons, the specific

Genotype

Tasks (ψ)	Solution (τ'_ψ)				Order (O)	Robot Assignment (ξ)
ψ_1	σ_2	σ_5	σ_1	σ_3	4	r_3
ψ_2	σ_6	σ_2			3	r_2
ψ_3	σ_1	σ_2	σ_1		1	r_1
ψ_4	σ_3	σ_2	σ_1	σ_4	2	r_3
ψ_5	σ_5	σ_1			5	r_1

Phenotype

Robot	Tasks along with order)	Tasks (sorted by order)	Trajectory (before DP fusion)	Trajectory (after DP fusion)
r_1	(ψ_3,1), (ψ_5, 5)	ψ_3, ψ_5	$[\sigma_1, \sigma_2, \sigma_1] \oplus_D [\sigma_5, \sigma_1]$	$S_1, \sigma_1, \sigma_2, \sigma_5, \sigma_1, \sigma_1$
r_2	(ψ_2, 3)	ψ_2	$[\sigma_6, \sigma_2]$	S_2, σ_6, σ_2
r_3	(ψ_1, 4), (ψ_4, 2)	ψ_4, ψ_1	$[\sigma_3, \sigma_2, \sigma_1, \sigma_4] \oplus_D [\sigma_2, \sigma_5, \sigma_1, \sigma_3]$	$S_3, \sigma_3, \sigma_2, \sigma_2, \sigma_5, \sigma_1, \sigma_1, \sigma_3, \sigma_4$

Fig. 2 Individual representation and conversion from genotype to phenotype. Shaded regions are frozen for optimization

operation that contributed at the end also needs to be stored specifically and hence let $\delta(i,j)$ be the specific site where the robot is located at the end in the optimal calculation of $d(i,j)$. Using the principles of Dynamic Programming, the optimal fusion is given by Algorithm 1.

Lines 1–7 are used to initialize the Dynamic Programming memorization table for the unit case for the cost (d), parents (π), and last operation (δ). In this algorithm, the Dynamic Programming is 2-dimensional since it is parametrized by two variables (say i and j). The unit case of the 2-dimensional Dynamic Programming is hence when $i = 0$ (and similarly another when $j = 0$). This is a 1-dimensional Dynamic Programming problem. This problem is referred to as initialization because the 1-dimensional Dynamic Programming is an initialization to the main 2-dimensional Dynamic Programming problem. If the second input string is empty, the solution of fusion till the ith character of the first input string is the first string till the ith character (with the corresponding parent) whose cost is computed and stored. Similarly, for the case when the first input string is empty.

Lines 8–15 form the main logic of Dynamic Programming. All sub-problems are solved from smaller to larger. Line 10 decides for the solution of $d(i,j)$ whether the sub-problem d(i − 1,j) is a better candidate or the sub-problem d(i,j − 1), given that both decompositions are possible. Leaving the frozen conditions, the line states that if adding the character $\tau^{\text{task}}_{\psi_1}(i)$ to the sub-problem, $d(i-1,j)$ is better than adding the character $\tau^{\text{task}}_{\psi_2}(j)$ to the sub-problem d(i − 1,j), the first sub-problem will be used and vice versa. The frozen constraints additionally ensure that if one of the two characters ($\tau^{\text{task}}_{\psi_1}(i)$ or $\tau^{\text{task}}_{\psi_2}(j)$) is frozen while the other is not, the frozen character takes priority. Lines 11–12 store the result when the sub-problem d(i − 1,j) is better, and lines 14–15 store the result when the sub-problem d(i,j − 1) is better.

In lines 16–21, p iterates over all the sub-problems that lead to the solution for the overall problem d(len($\tau^{\text{task}}_{\psi_2}(j)$), len($\tau^{\text{task}}_{\psi_2}(j)$)). As it iterates over all the sub-problems, the solution is stored in τ. If the parent points to the first sub-problem, the corresponding character from the first input string is added at the beginning of the soliton τ and vice versa.

Algorithm 1: DP Operator $\oplus_D\left(\boldsymbol{\tau}_{\psi_1}^{task}, \boldsymbol{\tau}_{\psi_2}^{task}\right)$

```
    //Initialization
1.  $d(0,0) \leftarrow 0, \pi(0,0) \leftarrow 0, \delta(0,0) \leftarrow S_r$
2.  for $i$ from 1 to $len(\tau_{\psi_1}^{task})$
3.      $d(i,0) \leftarrow d(i-1,0) + c\left(\delta(i-1,0), \tau_{\psi_1}^{task}(i)\right)$
4.      $\pi(i,0) \leftarrow 1, \delta(i,0) \leftarrow loc\left(\tau_{\psi_1}^{task}(i)\right)$
5.  for $j$ from 1 to $len(\tau_{\psi_2}^{task})$
6.      $d(0,j) \leftarrow d(0,j-1) + c\left(\delta(i,j-1), \tau_{\psi_2}^{task}(j)\right)$
7.      $\pi(0,j) \leftarrow 2, \delta(0,j) \leftarrow loc\left(\tau_{\psi_2}^{task}(j)\right)$
    //Dynamic Programming
8.  for $i$ from 1 to $len(\tau_{\psi_1}^{task})$
9.      for j from 1 to $len(\tau_{\psi_2}^{task})$
10.         if $\neg frozen\left(\tau_{\psi_1}^{task}(i-1)\right) \wedge frozen\left(\tau_{\psi_2}^{task}(j-1)\right) \vee$
            $\neg\left(frozen\left(\tau_{\psi_1}^{task}(i-1)\right) \wedge \neg frozen\left(\tau_{\psi_2}^{task}(j-1)\right)\right) \wedge$
            $d(i-1,j) + c\left(\delta(i-1,j), \tau_{\psi_1}^{task}(i)\right) < d(i,j-1) + c\left(\delta(i,j-1), \tau_{\psi_2}^{task}(j)\right)$
11.             $d(i,j) \leftarrow d(i-1,j) + c\left(\delta(i-1,j), \tau_{\psi_1}^{task}(i)\right)$
12.             $\pi(i,j) \leftarrow 1, \delta(i,j) \leftarrow loc\left(\tau_{\psi_1}^{task}(i)\right)$
13.         else
14.             $d(i,j) \leftarrow d(i,j-1) + c\left(\delta(i,j-1), \tau_{\psi_2}^{task}(j)\right)$
15.             $\pi(i,j) \leftarrow 2, \delta(i,j) \leftarrow loc\left(\tau_{\psi_2}^{task}(j)\right)$
    //Tracing the path
16. $\tau \leftarrow \emptyset, p \leftarrow \left(len(\tau_{\psi_1}^{task}), len(\tau_{\psi_2}^{task})\right)$
17. while $p(1) > 0 \vee p(2) > 0$
18.     if $\pi(p(1), p(2)) = 1$
19.         $\tau \leftarrow [\tau_{\psi_1}^{task}(p(1)), \tau], p \leftarrow [p(1)-1, p(2)]$
20.     else
21.         $\tau \leftarrow [\tau_{\psi_2}^{task}(p(2)), \tau], p \leftarrow [p(1), p(2)-1]$
22. return τ
```

The unit operations (e.g. collecting an item, getting a document signed, etc.) are exclusive and the robot cannot do two unit operations at the same time. The solution to a task is performing multiple unit operations sequentially. There are synergies between the solution of two tasks, that is the robot will perform multiple steps of different tasks interchangeably. This is primarily done by using Dynamic Programming.

As an example, consider that sequentially solving $[\sigma_1, \sigma_2, \sigma_3]$ is a valid solution of the first task, while sequentially solving $[\sigma_a, \sigma_b]$ is a valid solution of the second task. Here, all σ_i are unit operations, while these are two of the many tasks for a robot. Now the valid solutions for simultaneously solving the two tasks are $[\sigma_a, \sigma_b, \sigma_1, \sigma_2, \sigma_3]$, $[\sigma_a, \sigma_1, \sigma_b, \sigma_2, \sigma_3]$, $[\sigma_a, \sigma_1, \sigma_2, \sigma_b, \sigma_3]$, $[\sigma_a, \sigma_1, \sigma_2, \sigma_3, \sigma_b]$, $[\sigma_1, \sigma_a, \sigma_b, \sigma_2, \sigma_3]$, $[\sigma_1, \sigma_a, \sigma_2, \sigma_b, \sigma_3]$, $[\sigma_1, \sigma_a, \sigma_2, \sigma_3, \sigma_b]$, $[\sigma_1, \sigma_2, \sigma_a, \sigma_b, \sigma_3]$, $[\sigma_1, \sigma_2, \sigma_a, \sigma_3, \sigma_b,]$, $[\sigma_1, \sigma_2, \sigma_a, \sigma_b, \sigma_3]$, $[\sigma_1, \sigma_2, \sigma_a, \sigma_3, \sigma_b]$, and $[\sigma_1, \sigma_2, \sigma_3, \sigma_a, \sigma_b]$. In all these the robot is switching between the solution of the first task $[\sigma_1, \sigma_2, \sigma_3]$ and the second task $[\sigma_a, \sigma_b]$. Dynamic Programming selects the best combination out of all these combinations, while Genetic Algorithm selects good solutions ($[\sigma_1, \sigma_2, \sigma_3]$ and $[\sigma_a, \sigma_b]$) that after the application of Dynamic Programming become the best solution.

The frozen variables have already been operationalized and therefore should be used before any non-frozen variable in the Dynamic Programming fusion. In other words, a new task cannot come in between the solution of a task that has already been performed and frozen. A requirement is that the linear plan $T' = \{\tau'_1, \tau'_2, \tau'_3 \ldots \tau'_z\}$ should facilitate mapping to every detail available in the genotype and hence every element in the plan and hence all variables used in the Dynamic Programming, including those coming from the task plan τ'_ψ are annotated with their frozen status, robot assigned, a task that the variable is a part of.

Here care must be taken in the implementation of S_r as the source of the robot. It cannot be taken as the current position, since the individual also encodes the operations already performed. It cannot be the initial source since no tasks are available at the time the robot starts, and those are incrementally added. Till a robot has some frozen tasks it does not matter which source is taken since the frozen parts are the same throughout individuals which produce the same fusion whose cost is a constant addition. The source for a task is the robot's position at the end of its current operation, at the time the task is inserted, which needs to be updated if a robot has no operations frozen.

Let $\mathrm{T}_{\text{task}} = \left\{\tau_1^{\text{task}}, \tau_2^{\text{task}}, \tau_3^{\text{task}} \ldots, \tau_n^{\text{task}}\right\}$ be the trajectory corresponding to the tasks as stored in the genotype. The fusion is between multiple tasks for a robot and hence the fused sequence is given by Eqs. (16) and (17), and the cost of fusion is given by Eq. (18)

$$\tau'_r = \tau_{O(1)}^{\text{task}} \oplus_D \tau_{O(2)}^{\text{task}} \oplus_D \tau_{O(2)}^{\text{task}} \cdots \oplus_D \tau_{O(\text{len}(\tau'_r))}^{\text{task}} \tag{16}$$

$$\tau'_r = \oplus_{D,i=1}^{n}\left(\tau_{O(i)}^{\text{task}}\right),\ \xi(\psi = O(i)) = r \tag{17}$$

$$\text{cost}_r^{\text{task}}(\mathrm{T}_{\text{task}}, O, \xi) = C\left(\tau'_r\right) \tag{18}$$

$\text{cost}_r^{\text{task}}(\mathrm{T}_{\text{task}}, O, \xi)$ denotes the cost of the task plans T_{task} with the order of fusion O and robot assignment function ξ for the robot r. $C(\tau'_r)$ is the cost of the sequence of operations τ'_r. The associativity is strictly from left to right and the order of taking the tasks is as per the order specified in the evolutionary individual denoted by the $O()$ subscripts in the notation. For each robot, a penalty is added in proportion to the number of tasks that are assigned to a robot but not solved by the robot as per the evolutionary individual. The magnitude by which the robot misses to solve the mission is also coded if the verification engine reports the same. The robots have problems in doing too many operations, and therefore, a small penalty is added for smaller solutions (in terms of length and not cost). Since optimality is of concern, the individual is also assessed and if a task is satisfied by the initial part of the individual only and all genes are not needed to satisfy the task, the string representing the solution for the task is trimmed. This acts as a local delete evolutionary operator.

The fitness function is the maximum cost among all robots given by Eq. (19).

$$\begin{aligned}\text{Fitness}\,(\mathrm{T}_{\text{task}}, O, \xi) = \max_r\ \text{cost}_r^{\text{task}}\,(\mathrm{T}_{\text{task}}, O, \xi) \\ + \alpha\ \Sigma_r \text{cost}_r^{\text{task}}\,(\mathrm{T}_{\text{task}}, O, \xi) \\ + \beta\ \Sigma_\psi\ \text{penalty}\left(\tau_\psi^{\text{task}}\right) \\ + \gamma\ \Sigma_\psi \text{len}\left(\tau'_r\right)\end{aligned} \tag{19}$$

Here α is a small constant for the summation factor. $\beta >> 1$ is the penalty constant and penalty($\tau^{\text{task}}{}_\psi$) function return the magnitude by which τ_ψ^{task} violates ψ. The penalty function can be adapted as per the language. As an example, if the task ψ asks the robot to sequentially do several complex operations (atomic operations with Boolean constraints), however, the robot only a few of them, then penalty $\left(\tau_\psi^{\text{task}}\right)$ is the number of operations in the sequence that were not performed by the robot. γ is a small constant to penalize too many operations and len$\left(\tau_\psi^{\text{task}}\right)$ is the length of the solution of the task.

The fitness function can thus be seen to be the cost of the robot working for the maximum time, with a small contribution coming from the other robots. There are penalties for not completely solving a task and for having a robot do too many operations. The fitness function is primarily optimized by changing the sequence of operations that solve a task $\left(\tau_\psi^{\text{task}}\right)$ and by changing the robot assigned to tasks (ξ). Trying all combinations of all robot operations can be computationally expensive. Hence, the planner considers every robot separately. The planar first notes the list of tasks assigned to the robot. The planer then considers all the operations associated with every task. Every task gives one sequence of operations. The sequences of all operations in all tasks are integrated into one sequence by using Dynamic Programming. This gives the sequence of operations (along with cost) of every robot, used for the fitness calculation. The evolutionary algorithm optimizes this fitness function.

This also points to the relation between evolution and Dynamic Programming. Dynamic Programming is an integrator that integrates the task sequence of operations to produce a sequence of operations to be performed by the robot. The evolution hence optimizes the sequence of operations to solve a task and not the sequence of operations to be solved by the robot, which is an easier problem. Informally, Dynamic Programming may be thought of as a local search applied to every individual in every generation.

5.4 Dynamic programming based diversity preservation

Since the tasks are continuously inserted, frozen, and deleted from the mission specification, diversity preservation is important to use. More generally, the optimization problem is against a dynamically changing fitness landscape, or the optimization is for a problem whose fitness function keeps changing due to the change in the mission specifications. A conventional optimization algorithm aims to get convergence over time, possibly converging into a global optimum. However, diversity preservation techniques add constraints that make it less likely to delete an individual that adds to diversity by being a member of an under-represented area. This significantly delays or avoids convergence. If the fitness function is dynamic, a global optimum may later become a local optimum, and a local optimum may later become a global optimum. For example, it is possible that a sub-optimal plan later becomes optimal due to the insertion of a competing task in the system by the user. The convergence into the current global optima is thus not preferred since a change in the fitness function may make the problem converged into what will later turn out to be a local optimum. The aim is now to keep enough individuals spread across the fitness landscape, such that a change in the fitness function means that the nearby individuals can quickly adapt and change the evolutionary pressures to attract more individuals towards the current global optima. Diversity preservation is hence useful to maintain a diverse spread of individuals in anticipation of a change in the fitness landscape.

The crowding-based diversity preservation is used wherein the child is generated by using genetic operators, however, the child replaces the closest of the sampled set of parents, which restricts the population diversity to drop significantly.

The implementation of diversity preservation is straightforward. The problem associated is the absence of a distance measure since the two individuals in both populations have different sizes. Let $d^{\text{plan}}(X,Y)$ be the distance function to be made with X and Y as the 2 individuals of the population pool. Dynamic Programming is used to compute the distance between X and Y for the solution of the task ψ and the results are added for all possible tasks. A problem is that the distance between a solution of a task with itself should be zero, while the Dynamic Programming will return the cost of the same task as the cost of fusion, and hence the cost of the maximum of the two tasks is subtracted. The Dynamic Programming-based distance function is hence given by Eq. (20).

$$d^{\text{plan}}(X,Y) = \Sigma_{\psi}\left(C\left(\tau^{\text{task}}_{\psi,X} \oplus_D \tau^{\text{task}}_{\psi,Y}\right) - \max\left(C\left(\tau^{\text{task}}_{\psi,X}\right), C\left(\tau^{\text{task}}_{\psi,Y}\right)\right)\right) \tag{20}$$

Here $\tau^{\text{task}}_{\psi,X}$ and $\tau^{\text{task}}_{\psi,Y}$ is the solution (sequence of operations) of task ψ in X and Y, respectively. $\tau^{\text{task}}_{\psi,X} \oplus_D \tau^{\text{task}}_{\psi,Y}$ is the fusion of the two solutions by using Dynamic Programming and $C()$ function returns the cost of a solution (sequence of operation).

5.5 Freeze aware genetic operators

The Genetic Operators consist of crossover and mutation, and they need to be carried out for all the three variables used, which are task solutions $\left(\tau^{\text{task}}_{\psi}\right)$, order ($O$), and robot assignment (ξ). Typicality is that the operations must only alter the variables that are not frozen, which is a constraint easy to code in both crossover and mutation. Further, the crossover of the task solutions $\left(\tau^{\text{task}}_{\psi}\right)$ is interesting since the parents will be of a non-uniform size. However, crossover between parents of different sizes can be done heuristically as per literature available.

Two crossover techniques are used. The first is a scattered crossover applied only to non-frozen parts. The second exploits (non-strictly) combinatorial optimization characteristics of the algorithm in which case out of unfrozen parts, a random crossover point is taken like the 1-point crossover. Till the crossover point, the strings are directly taken and swapped, and inserted in the other part only if the count of all operations does not exceed the original count. The fundamental is to keep the count of operations of the children the same as the corresponding parent, and therefore, the leftover operations are randomly inserted to maintain the counts.

Mutations are also of 2 types. The structural mutation operations consist of deleting an operation or adding a new operation out of var(ψ). Here, var(ψ) is a collection of atomic operations that appear in the specification of the task ψ. The parametric mutation operations incorporate swapping two operations and replacing an operation with a random one in var(ψ). The crossover and mutation of order variables (O) is a simple combinatorial optimization, and the robot assignment (ξ) is a simple parametric optimization.

Local optimization in a master–slave memetic architecture is also applied since many times a good solution should be readily available. Every individual after every few generations is subjected to local optimization. If an individual is reasonably near to an optimum, the local optimization can readily place it very close to the optimum that will take quite some time for the evolutionary algorithm. The local optimization thus helps in getting good individuals earlier in the optimization process. The evolutionary algorithm benefits from the improved fitness of the individuals by the local optima that makes it easier to distinguish between the global and local optima for guiding convergence. The fitness function is additionally stored as a hash table so that the re-occurrence of the same individual does not result in computation of a new fitness function by a costly Dynamic Programming fusion and can be obtained immediately.

5.6 Reaching a mission site

Suppose a robot is operating on a plan τ'_r and the robot just completed an operation $\tau'_r(i)$ in the plan. Now the robot needs to benefit from the optimizations carried and incorporate any addition of tasks that happened while it was completing the operation $\tau'_r(i)$. Hence, the best-optimized plan at the time of completion of $\tau'_r(i)$ is loaded to the robot for performing the next task. The optimal sequence loaded by the optimizer will consist of the entire list of operations, including those that are already over, and therefore in the sequence, the first unfrozen operation is selected given by Eq. (21).

$$\text{nextOp}(r) = \tau'_r(j) : \neg\text{frozen}\left(\tau'_r(j)\right) \wedge \forall_{k<j}\ \text{frozen}\left(\tau'_r(k)\right) \tag{21}$$

Here frozen($\tau'_r(j)$) returns whether $\tau'_r(j)$ is frozen or not. It may be emphasized that it is assumed that the path $\tau'_r(j)$ consists of a trace of whether it was frozen in the task that it came from, the task, and the robot associated with the task in the genetic individual. This is possible since all information is available when integrating using Dynamic Programming.

Since the next operation (nextOp) is transmitted to the robot and it cannot be preempted, the same needs to be frozen for optimization in all individuals. The same calculation is also performed at the task level. The robot is asked to perform the next operation of the task $\psi = \text{task}(\text{nextOp}(r))$. The best individual (best) has a task trajectory $\tau^{\text{task}}_{\psi,\text{best}}$. Suppose the task trajectory has been traced till the index nextTaskOp(ψ) given by Eq. (22).

$$\text{nextTaskOp}(\psi) = \neg\text{frozen}\left(\tau^{\text{task}}_{\psi,\text{best}}(j)\right) \wedge \forall_{k<j}\ \text{frozen}\left(\tau^{\text{task}}_{\psi,\text{best}}(k)\right) \tag{22}$$

Here task(nextOp(r)) returns the task corresponding to the action nextOp(r). nextTaskOp(ψ) is the index of the first unfrozen path in the same task. The trajectory $\tau'_{\psi,\text{best}}$ (1 : nextTaskOp(ψ)) is frozen and therefore is appended to all the individuals in the population. This ensures that every individual in the population on a compilation by Dynamic Programming produces the same trajectory that was traced by the robot. Similarly, the robot assigned to the task is also frozen with the current robot assigned (r), if not done already. The task population for any individual X is hence given by Eqs. (23–26).

$$\text{frozen}\left(\tau'_{\psi,\text{best}}(\text{nextTaskOp}(\psi))\right) = \text{true} \tag{23}$$

$$\tau^{\text{task}}_{\psi,X} = \left[\tau^{\text{task}}_{\psi,\text{best}}(1 : \text{nextTaskOp}(\psi)),\ \tau^{\text{task}}_{\psi,X}(k : \text{end})\right] \tag{24}$$

$$k = \neg\text{frozen}\left(\tau^{\text{task}}_{\psi,X}(k)\right) \wedge_{l<k} \left(\text{frozen}\left(\tau^{\text{task}}_{\psi,X}(l)\right) \vee \tau^{\text{task}}_{\psi,X}(l) = \text{nextOp}(r)\right) \tag{25}$$

$$\xi(\psi, x) = r,\ \text{frozen}(\xi(\psi, x)) = \text{true} \tag{26}$$

Here frozen($\xi(\psi,x)$) denotes whether the robot assignment on task ψ and individual x is frozen. An assumption in (23–26) is that there indeed is a next operation for the robot available and the equations do not hold good when the robot just performed the last operation for the task. In such a case since the entire task is done, the task needs to be deleted from the mission ($\chi \leftarrow \chi - \{\psi\}$), the task populations need to be deleted for all individuals X, that is $T_{\text{task},X} \leftarrow T_{\text{task},X} - \left\{\tau^{\text{task}}_{\psi,X}\right\}$, along with the order ($O_X \leftarrow O_X - \{O(\psi)\}$) and robot assignment ($\xi_X \leftarrow \xi_X - \{\xi_X(\psi)\}$). In the specific current implementation, the ordering is defined as integers from 1 to the number of tasks, and therefore, the ordering variables will have to be re-worked for the same consistency.

5.7 Inserting tasks

An easier operation is when a user adds a new task to the mission ($\chi \leftarrow \chi \cup \{\psi\}$). This is easy because the population is already in the format of solutions for the task and this insertion means inserting, in all individuals of the population, a new random string representing the solution of the new task. The language considered in this paper allows constructing feasible solutions by a simulation process described as follows. Hence, all the initial individuals are always feasible. Let var(ψ) be the operations that occur in the specification of the task ψ. Since there is no negation, all operations in the initial population generation and thereafter shall be sampled from var(ψ). Further, the architecture of the task specification is exploited to generate the initial population by taking a random option in the case of an OR operator, taking all options in case of AND operation and generating a random permutation of them, and taking temporal constraints in the same order for adherence. Note that so far, no initial population was ever generated, other than a blank population only. This is because this is the only mechanism by which random individuals are randomly generated and added as solutions to the task incrementally.

5.8 Complexity

Since the overall algorithm is evolutionary, the complexity for the algorithm with a population of N individuals is given by a summation of each of the terms:

- $O(N \log N)$ for the sorting of all individuals as per their expectation values as a part of the selection operation (also for the selection).
- $O(N \times$ (fitness function complexity)) for the calculation of the fitness value of all individuals. Since the approach is not for any specific language, suppose $O(\text{verifier}(s,\psi))$ is the complexity of verification that the string s satisfies the task ψ in a language chosen by the user. The verification of every task requires $O(|\psi| \times \text{verifier}(s,\psi))$ complexity, where $|\psi|$ is the number of tasks. The fusion of the solution of two tasks requires an additional complexity of $O(\left|\tau_\psi^{\text{task}}\right|^2)$, where $|\tau_\psi^{\text{task}}|$ is the (maximum) length of the task solution. An iterative fusion of $|\psi|$ tasks see an increase in the fused string length and hence a complexity of $O(|\psi| \times |\tau_r| \times \left|\tau_\psi^{\text{task}}\right|)$, where $|\tau_r|$ is the (maximum) length of a robot plan. The maximum length of the robot plan is $|\psi| \times \left|\tau_\psi^{\text{task}}\right|$, which is the summation of all operations of a task, making the complexity for fusion as $O(|\psi|^2 \times \left|\tau_\psi^{\text{task}}\right|^2)$. The total complexity of computing all fitness values is thus $O(N \times |\psi| \times \text{verifier}(s,\psi) + N \times |\psi|^2 \times \left|\tau_\psi^{\text{task}}\right|^2)$. In our case, the verification algorithm consists of the AND, OR, and sequence operators with a complexity $O(\left|\tau_\psi^{\text{task}}\right| \times \text{len}(\psi) + \left|\tau_\psi^{\text{task}}\right|^2)$, where $\text{len}(\psi)$ is the length of the task specification string. This further gives the complexity as $O(N \times |\psi| \times \left|\tau_\psi^{\text{task}}\right| \times \text{len}(\psi) + N \times |\psi| \times \left|\tau_\psi^{\text{task}}\right|^2 + N \times |\psi|^2 \times \left|\tau_\psi^{\text{task}}\right|^2)$. Considering $\text{len}(\psi) \approx \left|\tau_\psi^{\text{task}}\right|$, the complexity is given by $O(N \times |\psi|^2 \times \left|\tau_\psi^{\text{task}}\right|^2)$
- $O(N \times \text{CF} \times |\psi| \times \left|\tau_\psi^{\text{task}}\right|^2)$ for diversity preservation where CF (crowding factor) is the number of individuals considered for replacement of the child population, $|\psi|$ is the number of tasks, $\left|\tau_\psi^{\text{task}}\right|$ is the (maximum) length of the task solution. The additional factor of $\left|\tau_\psi^{\text{task}}\right|^2$ is for Dynamic Programming fusion and $|\psi|$ is to fuse all the corresponding task solutions.

The total complexity is $O(N \log N + N \times |\psi| \times \textit{verifier}(s,\psi) + N \times |\psi|^2 \times \left|\tau_\psi^{\text{task}}\right|^2 + N \times \text{CF} \times |\psi| \times \left|\tau_\psi^{\text{task}}\right|^2)$ for the general case, and $O(N \log N + N \times |\psi|^2 \times \left|\tau_\psi^{\text{task}}\right|^2 + N \times \text{CF} \times |\psi| \times \left|\tau_\psi^{\text{task}}\right|^2)$ for the specific verification system considered. Practically considering $\text{CF} > \log N$ and $\text{CF} > |\psi|$, the complexity is given by $O(N \times \text{CF} \times |\psi| \times \left|\tau_\psi^{\text{task}}\right|^2)$.

It may be inquisitive that the number of robots does not appear in the complexity calculations, which is because the higher the number of robots, the smaller is the sequence length per robot and the smaller is the complexity. This means that the worst-case appears when the number of robots is the smallest that is used for the calculations. Even though the complexity reduces by an increase in the number of robots, adding robots creates new dimensions in the hypervolume of the search space of the evolutionary algorithm that makes the optimization a lot harder.

The approach is probabilistically optimal, like most techniques that use evolution, meaning that the probability of finding the optimal solution is non-zero. In other words, the approach is guaranteed to find an optimal solution as time tends to infinity. This happens because the mutation operator has the potential to suitably (and iteratively) modify any individual into an optimal solution, while the probability of survival of the individual by the selection operator is itself non-zero. It is not possible to compute a theoretical bound on the solution quality with respect to time as per the working of the algorithm.

6 Results

6.1 Comparative analysis

It is imperative to compare the algorithm with state-of-the-art approaches. First comparisons were aimed at using optimal planners. A naïve implementation of the planner could not solve a moderately sized problem because of the exponential time complexity. The Uniform Cost Search exhausted memory, while the Iterative Lengthening Algorithm was exhausted by time. Both searches are explained in [41]. An optimal search could only solve the problem of up to 13 variables. Thereafter, for comparisons, the optimality criterion was removed. The model verification approaches were used using the NuSMV library. The library could give non-optimal results till a problem size of 15 variables. The approach could not convert the LTL into an automaton for model verification because of exponential complexity. These are much smaller than the 24 tasks with up to 15 variables per task, totaling problems with 360 variables. Thereafter, heuristic searches were attempted that attacked the nature of the problem. Again, the search was primarily exponential to the number of variables and thus the approach could be used for problems with a small number of mission sites only. The observations are not surprising. One of the simplest cases of the proposed algorithm is to solve a Travelling Salesman problem, for which meta-heuristic evolutionary algorithms already have gained popularity, in contrast to all classic approaches. Unfortunately, there is no work on evolutionary algorithms for generic mission planning that can be used for comparisons.

The term 'number of variables' used in our approach is not the same as that used in the LTL. Assume a problem asking the robot to visit 5 places, say A, B, C, D, and E. In our methodology, the problem has 5 variables that were used to specify the problem. Let visit(A) be a propositional variable of the classic planning equivalent problem that can have a value of *true* or *false* at any point of time, and similarly for the other variables. The state-space consists of all possible values that all variables can take. Since visit(A) can take 2 values and similarly for all the other variables, the total state space has a size 2^5 or 32 (instead of 5). So, for n variables mentioned in our definition, the equivalent state space size for a classic planning system could go up to 2^n. LTL in the worst case can generate as many states as in the classic planning, using additional states for handling the temporal constraints. Hence directly comparing the term 'number of variables' for the proposed approach and the same term in LTL or classic planning systems is not valid. Different operators have different ways in which they affect the working of the algorithm. The OR operator makes the problem a lot easier as compared to the sequence, which is reasonably easier as compared to the AND operator. The results are with minimal use of the OR operator and sequence operator, and with maximum use of the AND operator.

The algorithm hence is compared with a baseline Genetic Algorithm that is the same as the proposed approach, however, optimizes the fusion to produce a linear path for the robots instead of using Dynamic Programming. This algorithm is henceforth called "*without Dynamic Programming*", a short form of the Genetic Algorithm optimization without using Dynamic Programming. A non-incremental variant of the proposed algorithm is also tested, henceforth called "*with dynamic Programming*", a short form of the non-incremental Genetic Algorithm optimization also using Dynamic Programming. The *proposed* algorithm is an incremental Genetic Algorithm optimization also using Dynamic Programming. The problem of comparing these algorithms with the proposed approach is that in the proposed approach, the robot keeps travelling and deletes the completed tasks, which changes the fitness function, while in other approaches, the robot only stays at the source till the optimal sequence is computed. As a result, the robot was not allowed to move in the proposed approach as well for analysis. The results were compared with random problems of a different number of tasks. Each task specification used 5 to 15 atomic operations in their specifications. The tasks were specified by using three operators which are AND ($\wedge$), OR ($\vee$), and sequencing (T). Since all operators are binary, the number of operators is 1 less than the number of operands (atomic operations). The toughest operator is the AND operator, which was generated with a probability of 0.7. The sequencing is simpler to optimize as the elements of the sequence may be solved independently and the results appended gives a reasonably good solution of the overall sequence. The probability of the use of sequencing was thus 0.2. OR is the simplest operator which rules out some atomic operations to be performed and hence the probability was kept as 0.1. To avoid the OR operation making the overall task very simple (say making the task as a simple operation OR a high complexity task), the OR operator was used within parenthesis and a very small number of operands, in the absence of sequence operator representing the conjunctive normal form. The problems involve 5 robots.

The comparative results for small and large problem sizes are shown in Fig. 3. Figure 3 also shows the costs relative to the optimal value. The proposed approach performed significantly better than GA. Under the setting, the incremental and non-incremental variants of the proposed algorithm become the same since the robot is not allowed to move. In the incremental version, the tasks are added one after the other. Typicality was that the incremental version performed minutely better consistently, meaning that it is better to optimize every task one by one and then add it to the population pool representing the entire population, which makes another advantage of the work. For the case with 4 tasks, the non-incremental version outperforms the incremental version since the problem is not complex and the non-incremental version allows generation of a new solution considering all the interaction between all tasks, while the incremental version attempts to initially build over the solutions with the existent tasks that can be somewhat restrictive. The total execution time may initially seem large. However, the tasks are complex, and the optimization happens as the robot performs. It was observed that every unit action involved the robot to approach and interact with the human which took an extremely long time in contrast to the times used for the experiments.

It is imperative to see how the algorithm behaves as the number of tasks is increased. The convergence obtained for all the discussed algorithms is shown in Fig. 4. As the number of tasks increases, the convergence gets slower. The larger be the number of tasks, the higher is the overall cost. Therefore, the cost is also plotted relative to the convergence optimal value. The proposed algorithm adds tasks one after the other. This gives it a very good starting point and gives it an overall early convergence. Even though increasing the number of tasks delays the convergence, it may also be seen that a major proportion of the convergence happens very quickly, for a reasonably high number of tasks. It is unlikely that the robot will do too many tasks simultaneously at the same time.

A better manner of understanding the scalability of the algorithm is to plot the convergence time against the increasing number of tasks. The plots for obtaining 80% of the convergence value and 85% of the convergence value are shown in Fig. 5. Generally, increasing the number of tasks increases the convergence time. However, sometimes adding

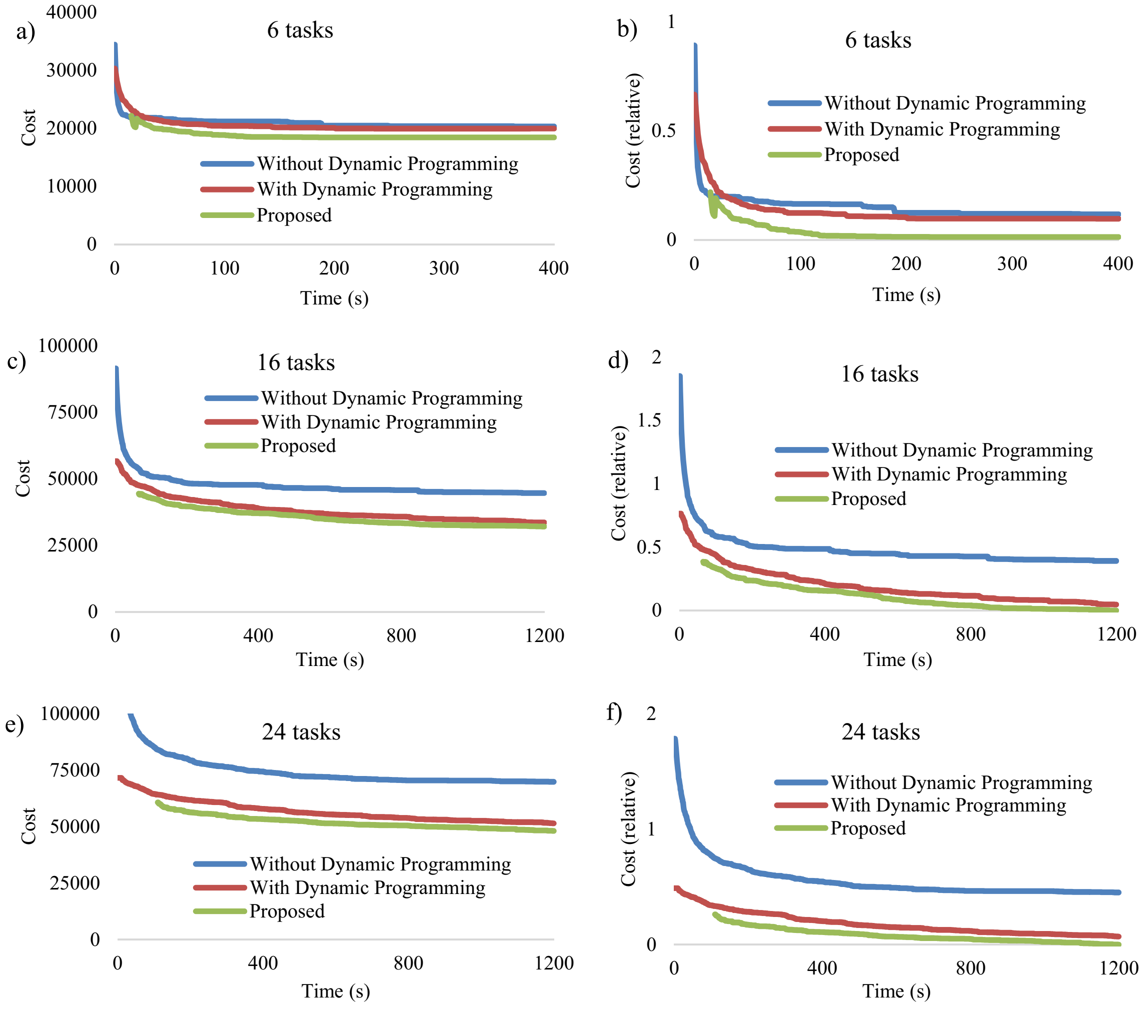


Fig. 3 Comparative results

a task makes the solver biased to get a solution that also simultaneously solves the other task. Therefore, sometimes the time decreases. It takes sufficiently more time to get 85% convergence in contrast to getting 80% convergence. The plots are only shown for a smaller number of tasks for the approach without dynamic programming since it is unable to get the set desirable value till the set timeout.

Finally, the convergent values reached by the different algorithms are shown in Fig. 6a. As the number of tasks increases, the robots must invest more traversal time. Figure 6b specifically shows the convergent costs relative to the converged optimal value. A value of 0 is therefore achieved by the best algorithm, while the closer the value is to 0, the better is the algorithm.

To understand the approach better, the algorithm is further compared against 7 different algorithms representing some greedy heuristics.

1. **Random Genetic Algorithm:** The first algorithm is a naïve implementation of a Genetic Algorithm that generates random strings, tests the validity of the strings, and computes the fitness function. The algorithm does not use any language-specific heuristic. After prolonged runs, it was observed that the algorithm could not generate any feasible solution for the mission.
2. **Random Search:** Another algorithm was used for the comparisons. The algorithm generates feasible solutions of tasks using the language-specific heuristics. The task solutions are randomly fused into a mission solution. There is no implementation of a Genetic Algorithm like an optimization algorithm. The algorithm only does random searches. The solutions generated using the approach were feasible due to the language-specific heuristics, but the solutions were extremely poor in terms of cost.
3. **Random Search with Dynamic Programming:** An attempt was made to improve the above algorithm by

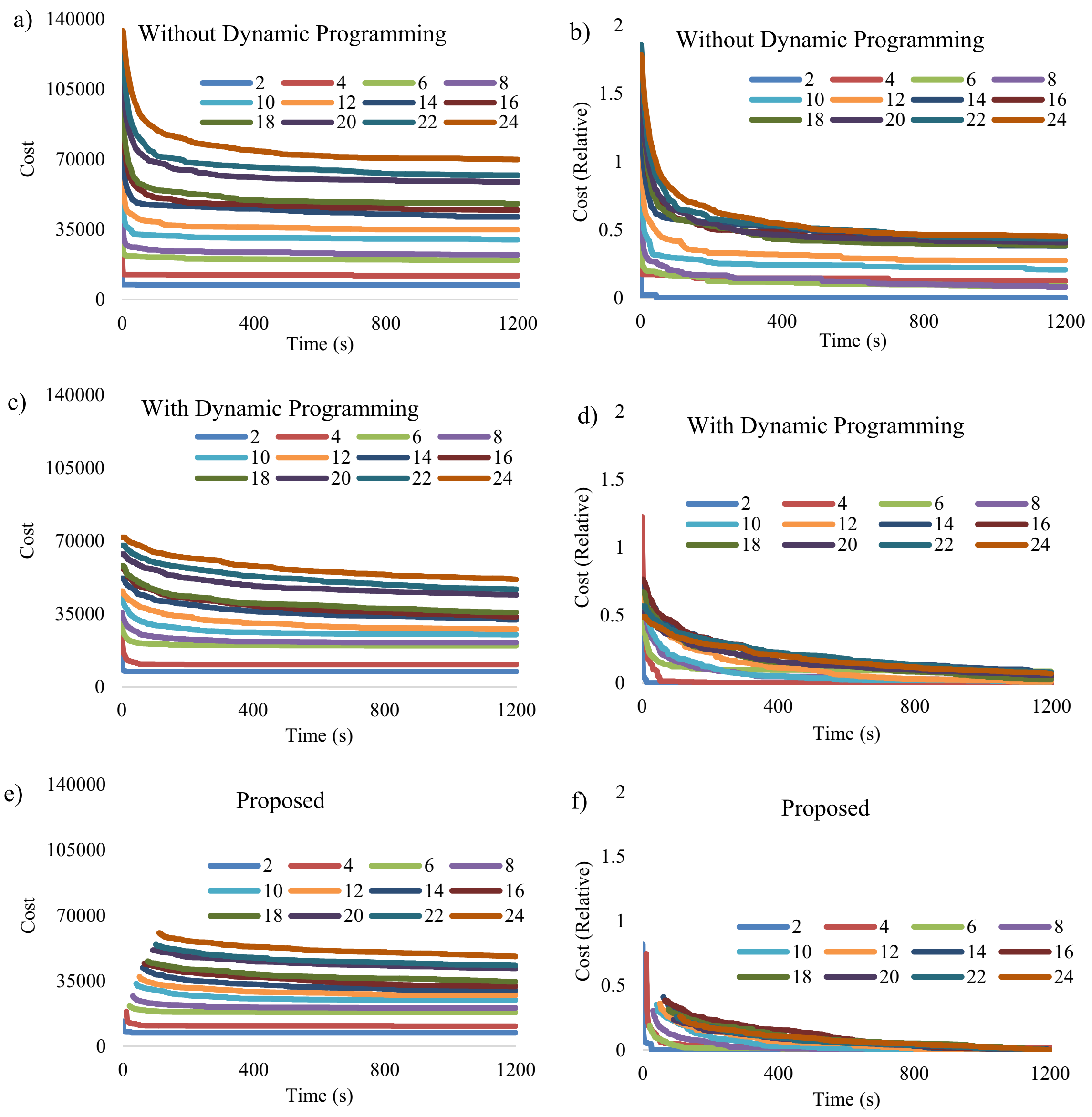


Fig. 4 Convergence with increasing the number of tasks

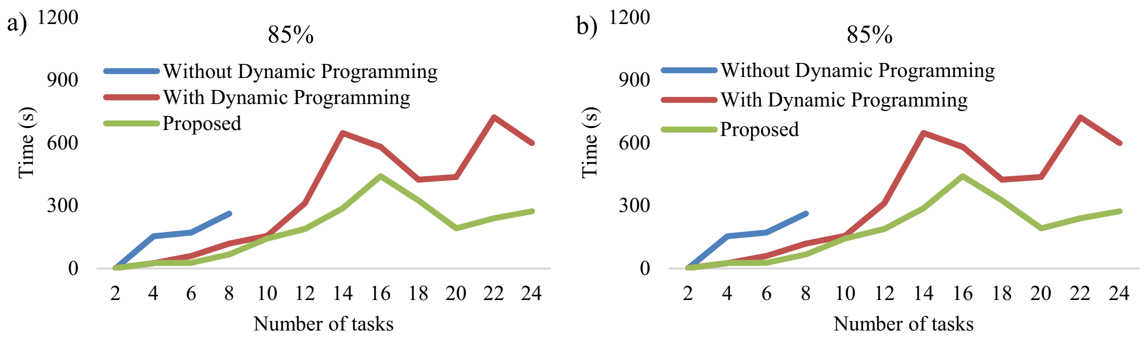
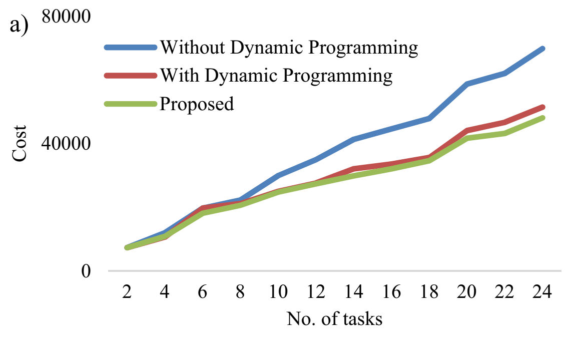


Fig. 5 Time to converge for different algorithms for an increasing number of tasks



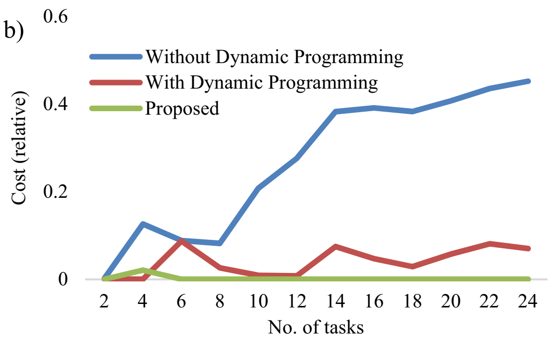


Fig. 6 Increased complexity with the number of tasks

Table 1 Comparative analysis of different approaches for different problem sizes

No. tasks	GA without DP[a]	GA with DP[b]	Proposed	Random GA[c]	Random search[d]	Random search with DP[e]	Greedy robot assign[f]	Seq solve tasks[g]	Greedy solver[h]	Greedy solver with DP[i]
2	7380	7371	7366	No Sol	8078	8078	*7353*	**7273**	11,328	11,328
4	12,043	**10,693**	*10,916*	No Sol	13,163	12,881	11,711	12,975	20,226	20,226
6	19,795	*19,772*	**18,185**	No Sol	25,498	24,447	22,708	21,148	32,433	29,343
8	22,341	*21,190*	**20,650**	No Sol	34,941	29,723	24,025	26,247	37,092	31,927
10	29,976	*25,035*	**24,814**	No Sol	47,684	36,538	29,890	34,380	48,609	36,892
12	34,996	*27,632*	**27,414**	No Sol	54,374	39,474	33,139	42,297	57,223	41,508
14	41,349	*32,137*	**29,907**	No Sol	69,788	45,905	36,100	49,703	67,589	46,106
16	44,665	*33,616*	**32,104**	No Sol	82,922	51,065	41,733	56,037	77,586	49,072
18	47,925	*35,653*	**34,654**	No Sol	89,154	53,317	42,978	58,757	82,062	48,873
20	58,731	*44,138*	**41,738**	No Sol	97,715	55,067	51,492	71,240	98,289	53,290
22	62,083	*46,748*	**43,255**	No Sol	112,530	61,087	57,530	76,598	104,738	56,161
24	69,892	*51,487*	**48,130**	No Sol	125,553	65,465	57,635	86,027	115,883	61,834

The best results are given in bold and the second best results are given in italics

[a]Genetic Algorithm approach that does not use Dynamic Programming for fusing task solutions

[b]Genetic Algorithm approach that uses Dynamic Programming for fusing task solutions, but all tasks are given at once rather than incrementally

[c]Genetic Algorithm that uses no language-specific heuristics

[d]Generates random individuals, feasible as per the language heuristics

[e]Generates random individuals to tasks, feasible as per the language heuristics, and fuses the task solutions using Dynamic Programming

[f]Assigns an incoming task to the robot that will finish its assigned tasks at the earliest while optimizing the overall solution

[g]The robots finish one task and only then attempt the next tasks. The task solutions are optimized

[h]Approach visits all sites in the same order as specified in AND, visits the closest site to the current site for OR, solves the tasks one after the other, and optimizes the robot assignment

[i]Approach visits all sites in the same order as specified in AND, visits the closest site to the current site for OR, uses the Dynamic Programming for fusing the task solutions, and optimizes the robot assignment

using Dynamic Programming for fusing the task solutions into the mission solutions. As expected, the results were significantly better in contrast to a random fusion. However, the solutions were still found to be reasonably poor in contrast to the proposed algorithm.

4. **Greedy Robot Assignment:** One of the aspects of the algorithm is a robot assignment problem that assigns the robot to the different tasks. An auctioning-based system is implemented. Upon getting a new task, the system asks for bids from all the robots and the best bid is given the task. The robots bid with the time of completion of their currently assigned tasks and the robot that is expected to complete all its currently assigned tasks at the earliest wins the bids. Genetic Algorithm is used for optimizing the task solutions and Dynamic Programming is used for the fusion of the task solutions. The approach performs well, however not as good as the proposed approach that shows the limitation of the heuristic.
5. **Sequentially Solving Tasks:** Several humans solve the missions by completing one person's order and only then accepting the other person's order. The task solutions are optimized. This heuristic is implemented, wherein a robot does not start a new task until and unless its current task

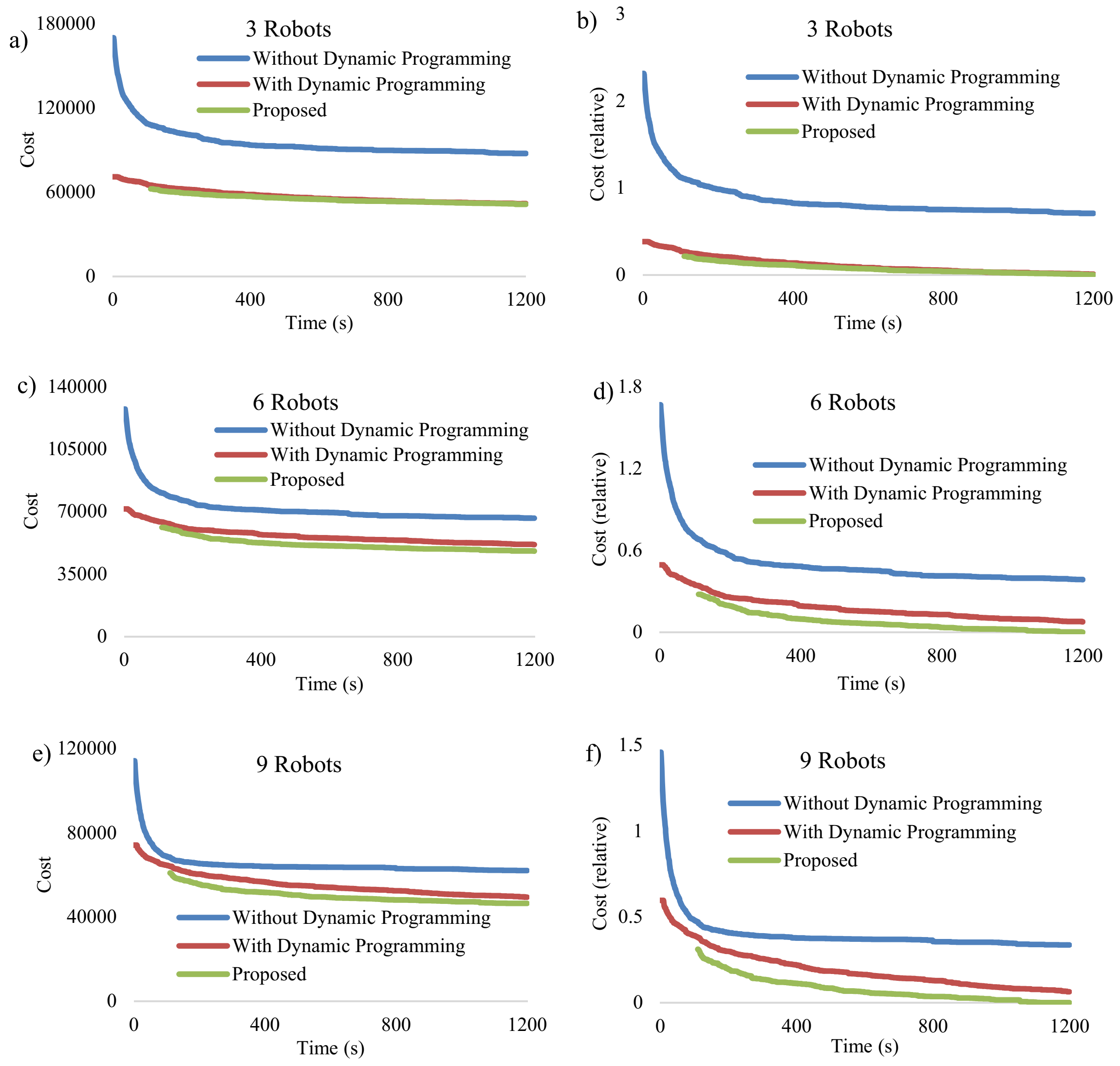


Fig. 7 Comparative results for different number of robots

is over. The heuristic is sub-optimal, and therefore, the approach generates very poor solutions.

6. **Greedy Solver:** A greedy solution to the problem is constructed. If the robot is asked to perform several operations using the AND operator, the robot will perform all of them in the same order as specified. If the robot is asked to perform one of the several operations using the OR operator, the robot would choose the lowest cost operation from its last location in the greedy solution, without caring about the future. For several tasks, the robot solves one task only after finishing an earlier one. The robot assignment is optimized. Essentially the approach does a greedy traversal. The heuristic is sub-optimal, and this results in paths that are as poor as a random traversal.
7. **Greedy Solver with Dynamic Programming:** The above algorithm was improved by using Dynamic Programming for fusing the task solutions. The approach resulted in better costs that was still reasonably poor due to the poor choice of greedy heuristics.

The results using the above methods are given in Table 1. Except for the very small problem size, the proposed approach performs better than all the other approaches.

6.2 Experiments with different numbers of robots

The mission planning problem is solved using several robots. The workplaces shall normally deploy many robots to efficiently solve the requirements of all the users. Therefore, the effect of increasing the number of robots is explicitly studied. The aim is also to see the betterment of the proposed algorithm in contrast to the naïve implementation of an evolutionary algorithm that does not use dynamic programming. The results for the different algorithms from small to

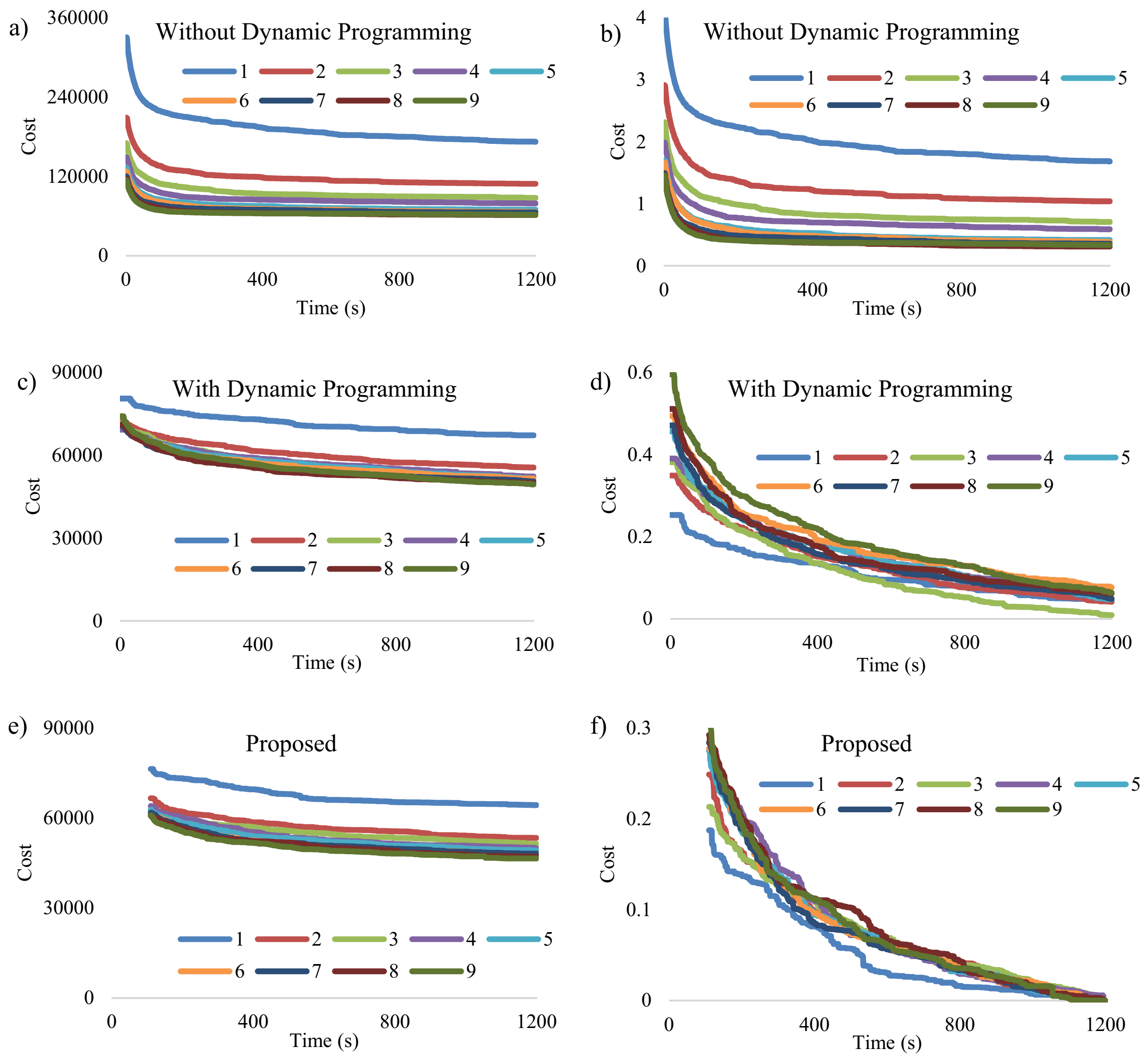


Fig. 8 Convergence with increasing the number of robots

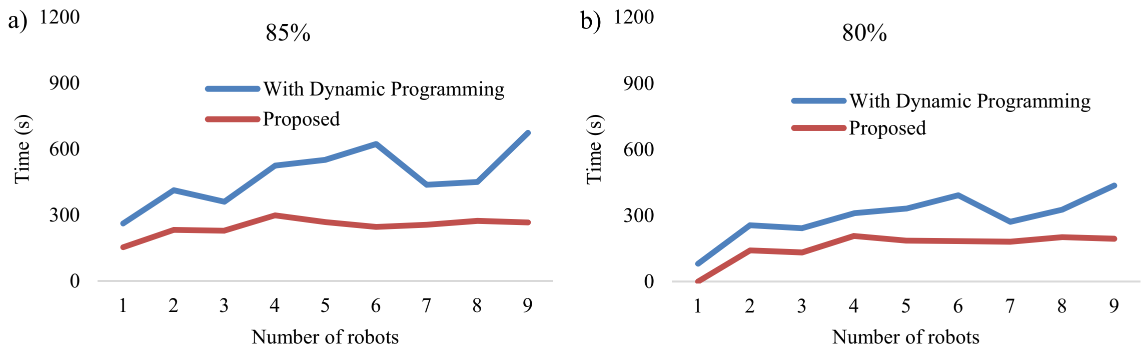


Fig. 9 Time to converge for different algorithms for an increasing number of robots

many robots are shown in Fig. 7. To make matters hard, the experiments are done using the maximum number of tasks investigated earlier, that is 24 tasks. The proposed approach outperforms the evolutionary approach without dynamic programming for all cases of varying the number of robots. Again, it is further observed that giving the tasks one after the other has a slight advantage to giving all tasks at once.

The effects of increasing the number of robots in different algorithms are further studied by plotting the convergence profiles. The results are shown in Fig. 8. In terms of the abso-

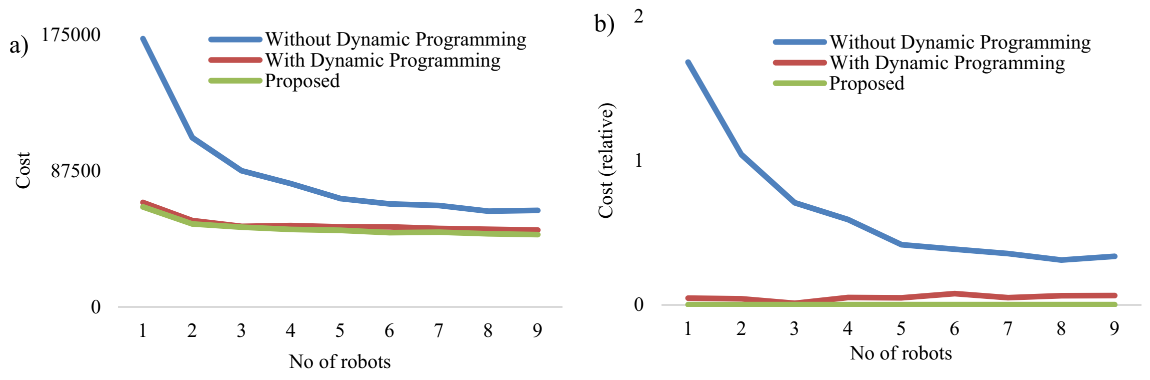


Fig. 10 Complexity with the number of robots

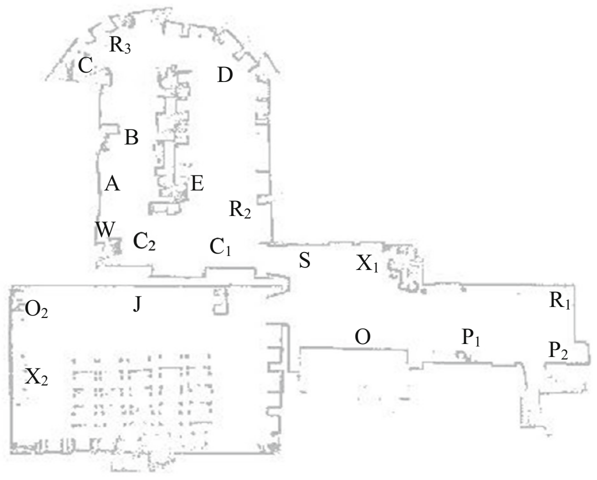


Fig. 11 Map of the robotics and machine intelligence laboratory along with the mission sites

lute values, increasing the number of robots decreases the overall cost, and therefore, the associated curves dig deeper. The convergence profiles however show no major change by increasing the number of robots. The relative costs more clearly show the effects of the number of robots. Increasing the number of robots generally makes it harder for the algorithms to converge. However, unlike increasing the number of tasks, in the case of robots, the change is not much profound.

The time to converge to 85% and 80% values are shown in Fig. 9. Initially, increasing the number of robots shows an appreciable change in the convergence time. However, later the convergence time becomes invariant to the increase in the number of robots. This happens when there is a sufficiently high number of robots to do the task, and adding more robots has a negligible improvement which has a negligible increase in the convergence time. The case of without Dynamic Programming is not shown as the method does not make the convergence to the specified limit value even after a prolonged time.

The effects of increasing the number of robots are further shown in Fig. 10. In terms of the absolute values, increasing the number of robots from 1 to 2 gives a significantly smaller cost. However, a still further increase in the number of robots has a much smaller reduction in the overall cost. The relative costs show the goodness of the different algorithms, with the proposed algorithm being the best and hence a relative cost of 0.

6.3 Experimental results

The results are obtained on a Pioneer LX robot, which is a differential wheel drive robot equipped with a lidar sensor for 2-dimensional vision and high accuracy wheel encoders for odometry. The robot has manufacturer-supported libraries that enable it to make a map of the workplace using the lidar sensor, encoders, and an onboard IMU. The robot was initially taken to the robotics facility at the Centre of Intelligent Robotics at the host institute to make a detailed map, while the robot was initially operated by using a joystick. The different places of interest were marked while the map was being made. The map was loaded into a manufacturer-provided map editor to mark the other places of interest. These places are used by the robot to convert a symbolic place name into real-world coordinates. The map was used to create a cost matrix between every pair of places. This was facilitated by using a Probabilistic Roadmap technique to fit a roadmap. The distance between every pair of places was computed by using a graph search algorithm. This cost matrix was loaded into the algorithm setup. The Robot Operating System (ROS) was used as a platform to allow easy integration between different modules. The robot is already ROS compatible with integration with the ROS navigation stack. This helps to pass coordinates of the places to the robot at the goal channel and the robot is autonomously moved to the place by using the ROS navigation stack and an acknowledgment is received on the channel. The robot uses the lidar, map, and wheel encoders to continuously localize

Box 1 Input mission statement

(i) A user realizes that he needs a robot to continue with his experiments. He calls the robot to his place (A), gives the robot the requisition and the robot should get the requisition signed by the secretary (S) and either of two professors (P_1 and P_2), before getting the equipment from the warehouse (W), notify the secretary by showing the robot (S) and then give the robot to the user at A. the task is given by $(\Diamond A)T(\Diamond S\wedge(\Diamond P_1\vee\Diamond P_2))T(\Diamond W)T(\Diamond S)T(\Diamond A)$

(ii) Another user is happy to be submitting his PhD thesis. So he calls the robot to take the thesis from him at B, get it signed by the supervisor (P_1) and submit it to the office (S). the task is given by $(\Diamond B)T(\Diamond P_1)T(\Diamond S)$

(iii) A user realizes that the robot he is working on is not working and therefore needs to be given for the repairs. Because it is a big equipment the report must be signed by the user (at C), any 2 witness (out of 4 friends A, B, D and E) present at the time, the secretary (S) and any of the two professors (P_1 and P_2). Thereafter the hardware is given to be secretary (S) for the official process. The task is given by $(\Diamond C)T((\Diamond A\wedge\Diamond B)\vee(\Diamond A\wedge\Diamond D)\vee(\Diamond A\wedge\Diamond E)\vee(\Diamond B\wedge\Diamond D)\vee(\Diamond B\wedge\Diamond E)\vee(\Diamond D\wedge\Diamond E))T(\Diamond S\wedge(\Diamond P_1\vee\Diamond P_2))T(\Diamond S)$

(iv) The meeting hall needs to be prepared that requires 2 new markers from any of the 2 cupboards (C_1 and C_2), water from any of the 2 outlets (O_1 and O2) turning on the projector (J). This should finally be reported to the secretary (S) and the requisitioner (P_1), who also collects the marker and water. The task is given by $(\Diamond C_1\vee\Diamond C_2)\wedge(\Diamond O_1\vee\Diamond O_2)\wedge(\Diamond J)T(\Diamond S\wedge\Diamond P_1)$

(v) The user D wants a printout from a common printer and asks the robot to get it from any of the common printers (X_1 and X_2), get D's signature on it and to submit it to any of the senior students (A, B or C) and finally to the professor (P_2). The task is given by $(\Diamond X_1\vee\Diamond X_2)T(\Diamond D)T(\Diamond A\vee\Diamond B\vee\Diamond C)T(\Diamond P_2)$

(vi) The user P_2 wants to check whether all his students A, C and E are present or not and sends a robot to physically check and report to him. The task is given by $(\Diamond A\wedge\Diamond C\wedge\Diamond E)T(\Diamond P_2)$

(vii) The user P_2 wants a coffee from any of the two outlets (O_1 and O_2) and a photocopy from any of the two machines (X_1 and X_2). The task is denoted by $((\Diamond O_1\vee\Diamond O_2)\wedge(\Diamond X_1\vee\Diamond X_2)T(\Diamond P_2)$

(viii) The user A also wants a coffee from any of the two outlets (O_1 and O_2). The task is given by $(\Diamond O_1\vee\Diamond O_2)T(\Diamond A)$

Box 2 Steps that were taken to solve the mission by the robots

Visit Order

(1) added task $(\Diamond A)T(\Diamond S\wedge(\Diamond P_1\vee\Diamond P_2))T(\Diamond W)T(\Diamond S)T(\Diamond A)$,

(2) R_3 heading to A for task $(\Diamond A)T(\Diamond S\wedge(\Diamond P_1\vee\Diamond P_2))T(\Diamond W)T(\Diamond S)T(\Diamond A)$,

(3) added task $(\Diamond B)T(\Diamond P_1)T(\Diamond S)$,

(4) R_3 heading to B for task $(\Diamond B)T(\Diamond P_1)T(\Diamond S)$,

(5) R_3 heading to P_1 for task $(\Diamond A)T(\Diamond S\wedge(\Diamond P_1\vee\Diamond P_2))T(\Diamond W)T(\Diamond S)T(\Diamond A)$,

(6) added task $(\Diamond C)T((\Diamond A\wedge\Diamond B)\vee(\Diamond A\wedge\Diamond D)\vee(\Diamond A\wedge\Diamond E)\vee(\Diamond B\wedge\Diamond D)\vee(\Diamond B\wedge\Diamond E)\vee(\Diamond D\wedge\Diamond E))T(\Diamond S\wedge(\Diamond P_1\vee\Diamond P_2))T(\Diamond S)$, **(7)** R_2 heading to C for task $(\Diamond C)T((\Diamond A\wedge\Diamond B)\vee(\Diamond A\wedge\Diamond D)\vee(\Diamond A\wedge\Diamond E)\vee(\Diamond B\wedge\Diamond D)\vee(\Diamond B\wedge\Diamond E)\vee(\Diamond D\wedge\Diamond E))T(\Diamond S\wedge(\Diamond P_1\vee\Diamond P_2))T(\Diamond S)$,

(8) R_2 heading to B for task $(\Diamond C)T((\Diamond A\wedge\Diamond B)\vee(\Diamond A\wedge\Diamond D)\vee(\Diamond A\wedge\Diamond E)\vee(\Diamond B\wedge\Diamond D)\vee(\Diamond B\wedge\Diamond E)\vee(\Diamond D\wedge\Diamond E))T(\Diamond S\wedge(\Diamond P_1\vee\Diamond P_2))T(\Diamond S)$,

(9) R_2 heading to D for task $(\Diamond C)T((\Diamond A\wedge\Diamond B)\vee(\Diamond A\wedge\Diamond D)\vee(\Diamond A\wedge\Diamond E)\vee(\Diamond B\wedge\Diamond D)\vee(\Diamond B\wedge\Diamond E)\vee(\Diamond D\wedge\Diamond E))T(\Diamond S\wedge(\Diamond P_1\vee\Diamond P_2))T(\Diamond S)$,

(10) R_2 heading to P_1 for task $(\Diamond C)T((\Diamond A\wedge\Diamond B)\vee(\Diamond A\wedge\Diamond D)\vee(\Diamond A\wedge\Diamond E)\vee(\Diamond B\wedge\Diamond D)\vee(\Diamond B\wedge\Diamond E)\vee(\Diamond D\wedge\Diamond E))T(\Diamond S\wedge(\Diamond P_1\vee\Diamond P_2))T(\Diamond S)$,

(11) added task $(\Diamond C_1\vee\Diamond C_2)\wedge(\Diamond O_1\vee\Diamond O_2)\wedge(\Diamond J)T(\Diamond S\wedge\Diamond P_1)$,

(12) R_1 heading to J for task $(\Diamond C1\vee\Diamond C2)\wedge(\Diamond O1\vee\Diamond O2)\wedge(\Diamond J)T(\Diamond S\wedge\Diamond P1)$,

(13) R_1 heading to C_2 for task $(\Diamond C_1\vee\Diamond C_2)\wedge(\Diamond O_1\vee\Diamond O_2)\wedge(\Diamond J)T(\Diamond S\wedge\Diamond P_1)$,

(14) R_1 heading to O_1 for task $(\Diamond C_1\vee\Diamond C_2)\wedge(\Diamond O_1\vee\Diamond O_2)\wedge(\Diamond J)T(\Diamond S\wedge\Diamond P_1)$,

(15) R_3 heading to P_1 for task $(\Diamond B)T(\Diamond P_1)T(\Diamond S)$,

(16) R_3 heading to S for task $(\Diamond A)T(\Diamond S\wedge(\Diamond P_1\vee\Diamond P_2))T(\Diamond W)T(\Diamond S)T(\Diamond A)$,

(17) R_1 heading to S for task $(\Diamond C_1\vee\Diamond C_2)\wedge(\Diamond O_1\vee\Diamond O_2)\wedge(\Diamond J)T(\Diamond S\wedge\Diamond P_1)$,

(18) R_2 heading to S for task $(\Diamond C)T((\Diamond A\wedge\Diamond B)\vee(\Diamond A\wedge\Diamond D)\vee(\Diamond A\wedge\Diamond E)\vee(\Diamond B\wedge\Diamond D)\vee(\Diamond B\wedge\Diamond E)\vee(\Diamond D\wedge\Diamond E))T(\Diamond S\wedge(\Diamond P_1\vee\Diamond P_2))T(\Diamond S)$,

(19) added task $(\Diamond X_1\vee\Diamond X_2)T(\Diamond D)T(\Diamond A\vee\Diamond B\vee\Diamond C)T(\Diamond P2)$,

(20) R_1 heading to X_1 for task $(\Diamond X_1\vee\Diamond X_2)T(\Diamond D)T(\Diamond A\vee\Diamond B\vee\Diamond C)T(\Diamond P_2)$,

(21) R_3 heading to W for task $(\Diamond A)T(\Diamond S\wedge(\Diamond P_1\vee\Diamond P_2))T(\Diamond W)T(\Diamond S)T(\Diamond A)$,

(22) R_2 heading to S for task $(\Diamond C)T((\Diamond A\wedge\Diamond B)\vee(\Diamond A\wedge\Diamond D)\vee(\Diamond A\wedge\Diamond E)\vee(\Diamond B\wedge\Diamond D)\vee(\Diamond B\wedge\Diamond E)\vee(\Diamond D\wedge\Diamond E))T(\Diamond S\wedge(\Diamond P_1\vee\Diamond P_2))T(\Diamond S)$,

(23) completed task $(\Diamond C)T((\Diamond A\wedge\Diamond B)\vee(\Diamond A\wedge\Diamond D)\vee(\Diamond A\wedge\Diamond E)\vee(\Diamond B\wedge\Diamond D)\vee(\Diamond B\wedge\Diamond E)\vee(\Diamond D\wedge\Diamond E))T(\Diamond S\wedge(\Diamond P_1\vee\Diamond P_2))T(\Diamond S)$,

(24) R_3 heading to S for task $(\Diamond B)T(\Diamond P_1)T(\Diamond S)$,

(25) completed task $(\Diamond B)T(\Diamond P_1)T(\Diamond S)$,

(26) added task $(\Diamond A\wedge\Diamond C\wedge\Diamond E)T(\Diamond P_2)$,

Box 2 continued

(27) R_1 heading to D for task $(\Diamond X_1 \vee \Diamond X_2)T(\Diamond D)T(\Diamond A \vee \Diamond B \vee \Diamond C)T(\Diamond P_2)$,
(28) R_3 heading to S for task $(\Diamond A)T(\Diamond S \wedge (\Diamond P_1 \vee \Diamond P_2))T(\Diamond W)T(\Diamond S)T(\Diamond A)$,
(29) R_3 heading to A for task $(\Diamond A)T(\Diamond S \wedge (\Diamond P_1 \vee \Diamond P_2))T(\Diamond W)T(\Diamond S)T(\Diamond A)$,
(30) completed task $(\Diamond A)T(\Diamond S \wedge (\Diamond P_1 \vee \Diamond P_2))T(\Diamond W)T(\Diamond S)T(\Diamond A)$,
(31) added task $((\Diamond O_1 \vee \Diamond O_2) \wedge (\Diamond X_1 \vee \Diamond X_2)T(\Diamond P_2)$,
(32) R_3 heading to C for task $(\Diamond A \wedge \Diamond C \wedge \Diamond E)T(\Diamond P_2)$,
(33) R_3 heading to A for task $(\Diamond A \wedge \Diamond C \wedge \Diamond E)T(\Diamond P_2)$,
(34) R_1 heading to A for task $(\Diamond X_1 \vee \Diamond X_2)T(\Diamond D)T(\Diamond A \vee \Diamond B \vee \Diamond C)T(\Diamond P_2)$,
(35) added task $(\Diamond O_1 \vee \Diamond O_2)T(\Diamond A)$,
(36) R_3 heading to E for task $(\Diamond A \wedge \Diamond C \wedge \Diamond E)T(\Diamond P_2)$,
(37) R_1 heading to P_1 for task $(\Diamond C_1 \vee \Diamond C_2) \wedge (\Diamond O_1 \vee \Diamond O_2) \wedge (\Diamond J)T(\Diamond S \wedge \Diamond P_1)$,
(38) completed task $(\Diamond C_1 \vee \Diamond C_2) \wedge (\Diamond O_1 \vee \Diamond O_2) \wedge (\Diamond J)T(\Diamond S \wedge \Diamond P_1)$,
(39) R_3 heading to O_1 for task $(\Diamond O_1 \vee \Diamond O_2)T(\Diamond A)$,
(40) R_3 heading to O_1 for task $((\Diamond O_1 \vee \Diamond O_2) \wedge (\Diamond X_1 \vee \Diamond X_2)T(\Diamond P_2)$,
(41) R_3 heading to X_1 for task $((\Diamond O_1 \vee \Diamond O_2) \wedge (\Diamond X_1 \vee \Diamond X_2)T(\Diamond P_2)$,
(42) R_3 heading to P_2 for task $((\Diamond O_1 \vee \Diamond O_2) \wedge (\Diamond X_1 \vee \Diamond X_2)T(\Diamond P_2)$,
(43) completed task $((\Diamond O_1 \vee \Diamond O_2) \wedge (\Diamond X_1 \vee \Diamond X_2)T(\Diamond P_2)$,
(44) R_1 heading to P_2 for task $(\Diamond X_1 \vee \Diamond X_2)T(\Diamond D)T(\Diamond A \vee \Diamond B \vee \Diamond C)T(\Diamond P_2)$,
(45) completed task $(\Diamond X_1 \vee \Diamond X_2)T(\Diamond D)T(\Diamond A \vee \Diamond B \vee \Diamond C)T(\Diamond P_2)$,
(46) R_3 heading to P_2 for task $(\Diamond A \wedge \Diamond C \wedge \Diamond E)T(\Diamond P_2)$,
(47) completed task $(\Diamond A \wedge \Diamond C \wedge \Diamond E)T(\Diamond P_2)$,
(48) R_3 heading to A for task $(\Diamond O_1 \vee \Diamond O_2)T(\Diamond A)$,
(49) completed task $(\Diamond O_1 \vee \Diamond O_2)T(\Diamond A)$

while using a fusion of a deliberative and a reactive planning technique for the motion. The physical motion happens by using a manufacturer-specific control module. The proposed algorithm computes an operation, consisting of the place to be visited by the robot and the operation to perform (currently limited to saying a few statements). The goal location is given to the goal channel and once an acknowledgment is received, the operation is performed. Since an audio message is the only operation now, the message is played on the robot's speaker, followed by a pre-specified time wait.

Consider the mission to be performed by 3 robots, initially placed at R_1, R_2, and R_3. The initial position of the robots along with the mission sites are shown in Fig. 11. Box 1 shows the different tasks given by different users in the order of input and Box 2 shows the output log of the robot as it completes all tasks step by step. The snapshots of the robot's execution are given in Fig. 12. The robots chose optimal sequences and completed all the tasks that were given. The tasks are added one after the other as per a pre-specified time in the simulator. Since there was only 1 available robot in the laboratory, the results were first recorded in the simulator under live settings, and the sequences were transferred to the robot. Further, since the actual robot was slow, there was enough time to highly optimize the plans, the results of which would not have been transferable to faster robots. Hence, the simulations purposefully assumed faster robots. The manipulation tasks were not done by the robot since it did not have a manipulator and those may be assumed to have been performed. Due to the heavy constraints of space, only one result is discussed.

7 Conclusions

In this paper, the problem of complex mission planning with multiple robots was attempted. Classic approaches work on generic temporal logic and typically have an exponential complexity and are not scalable to many variables. The proposed approach uses Boolean and sequencing specifications to specify a mission in a language that is generic enough to represent many problems of service robotics while guaranteeing a polynomial-time verification. The synthesis of a trajectory is thus done by using an evolutionary approach. The proposed solution is hence significantly better than the model verification approaches that have exponential complexity and are solvable for only a small number of mission sites.

The focus was to work incrementally, wherein the robot keeps optimizing the mission solution as it operates. This acts as a natural stopping criterion of the evolutionary mis-

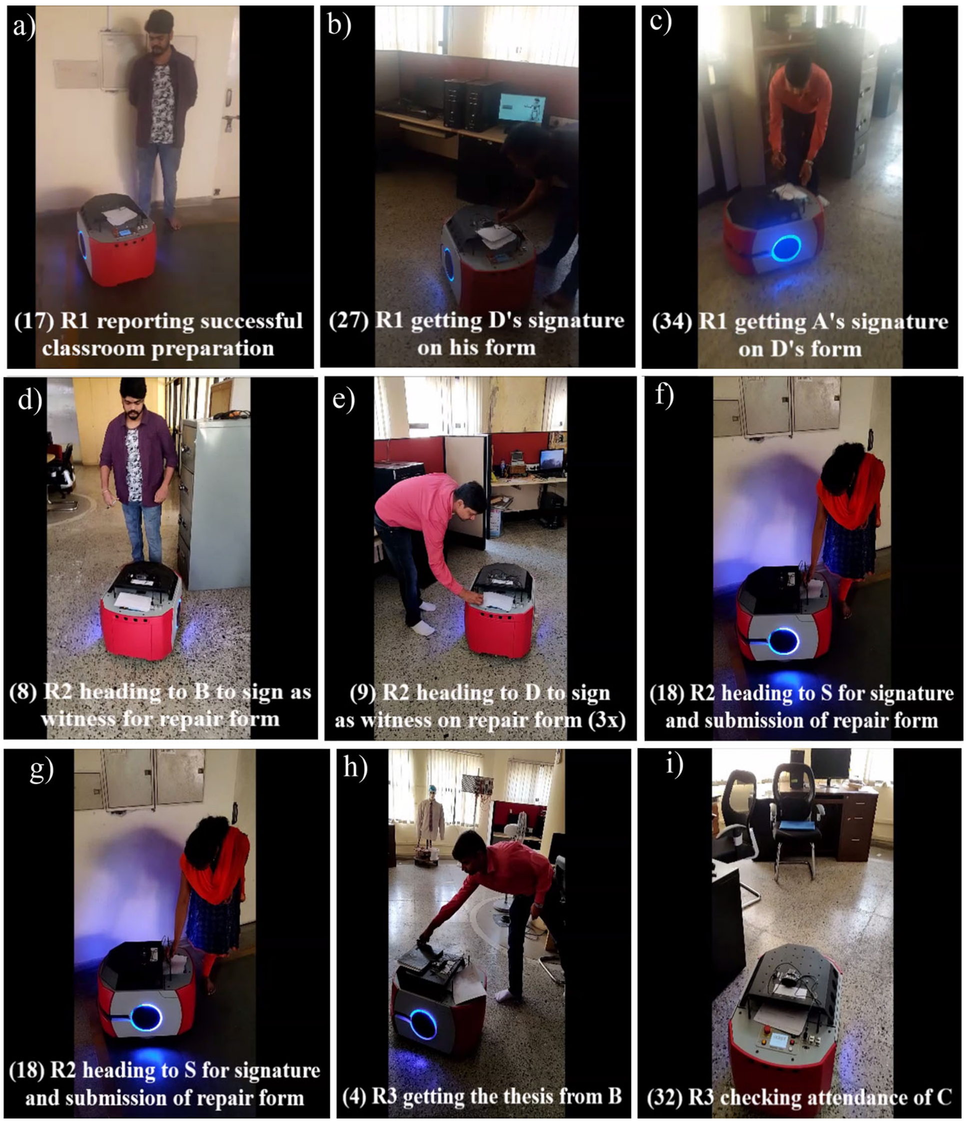

a)
(17) R1 reporting successful classroom preparation
b)
(27) R1 getting D's signature on his form
c)
(34) R1 getting A's signature on D's form
d)
(8) R2 heading to B to sign as witness for repair form
e)
(9) R2 heading to D to sign as witness on repair form (3x)
f)
(18) R2 heading to S for signature and submission of repair form
g)
(18) R2 heading to S for signature and submission of repair form
h)
(4) R3 getting the thesis from B
i)
(32) R3 checking attendance of C

Fig. 12 Experimental results. The first row has the execution of robot 1, the second row of robot 2, and the last row of robot 3

sion solver. Even though a lot of work has already been done for planning using evolutionary computation, most evolutionary approaches do not consider the incremental nature of the problem. The research has highlighted new issues with incremental evolution which were handled in the algorithm by freezing some genes in optimization and being conscious of the frozen variables in all aspects of the optimization. The problem is harder since the genotype is compiled to a different domain to give the physically traced trajectory. In non-incremental implementations, if a sub-optimal solution is found early, there is nothing that the robot does to optimize the plan while it is performing the mission. Further, the robot may have to wait for a prolonged time for a mission solution to be computed in the first place. The incremental implementations solve both the limitations.

It is important to understand that the verification-based approaches solve for the complete class of LTL, that is for all systems that can be specified using an LTL formulation. This paper does not solve the same problem. It cannot take any general LTL as an input and use approximations for solving the same problem. The approach is only for polynomial-verifiable languages, while the restriction that a string can be verified in a polynomial-time already means that many constraints shall have to be imposed. The experiments are done using a language that only uses AND, OR, and sequencing operators, which is a small subset of the entire representation capability of the LTL. While this is a major disadvantage, the paper argues that the language can cover a lot of scenarios specific to service robotics, where a robotic team services the everyday needs of humans. Attempts were made to quantify and measure the approximation against optimal solvers, by running the optimal solvers for a prolonged time. However, the optimal solvers having an exponential complexity could only give a result for problems of small size, while the exponential complexity suggests that waiting for a prolonged duration of time does not ensure a solution.

Since the approach marks a paradigm shift in the solution class to the mission planning problem, there were no competent methods for comparison available in the literature. Comparisons were hence shown on using a Genetic Algorithm as a baseline, and to make the baseline as strong as possible, every heuristic available in the problem and used by the solver was also added in the baseline Genetic Algorithm. Comparisons were also made with a non-iterative version of the proposed algorithm that took all tasks at once instead of taking them one by one as is in the case of the proposed algorithm. Comparisons were also made against several greedy heuristics that may be thought of for solving the problem. The proposed algorithm beats all these approaches. The system was tested on a real robot named Pioneer LX.

Acknowledgements This work is sponsored by the Science and Engineering Research Board, Department of Science and Technology through the Project Grant ECR/2015/000406 and the Indian Institute of Information Technology, Allahabad.

Author contributions Not applicable.

Availability of data and material (data transparency) Not applicable.

Code availability (software application or custom code) Not applicable.

Declarations

Conflict of interest Not applicable.

Appendix: Theorems and Proofs

Theorem 1 *The addition of a character in a string that satisfies task ψ consisting of the sequence operation only does not make the solution unsatisfiable.*

Proof Let the task be $\psi =$ "σ_0^ψ Then σ_1^ψ" The smallest string that satisfies the task is $s = [\sigma_0^\psi, \sigma_1^\psi]$. Let a be added to the string at any general location, giving the modified string as $s' = [\sigma_0^\psi, \sigma_1^\psi, a]$, or string $s' = [\sigma_0^\psi, a, \sigma_1^\psi]$, or string $s' = [a, \sigma_0^\psi, \sigma_1^\psi]$. Since both operations are done by the robot in the same sequence in all 3 options and (as per assumptions) the insertion of a is conflict-free to the operations σ_0^ψ and σ_1^ψ, the new string s' is still a valid solution to ψ. □

Theorem 2 *The addition of a character in a string that satisfies task ψ consisting of the AND ($\wedge$) operation only does not make the solution unsatisfiable.*

Proof Let the task be $\psi =$ "$\sigma_0^\psi \wedge \sigma_1^\psi$" The smallest string that satisfies the task is any permutation of σ_0^ψ and σ_1^ψ, say $s = [\sigma_1^\psi, \sigma_0^\psi]$ (and equivalently for $s = [\sigma_0^\psi, \sigma_1^\psi]$). Let a be added to the string at any general location, giving the modified string $s' = [\sigma_1^\psi, \sigma_0^\psi, a]$, or string $s' = [\sigma_1^\psi, a, \sigma_0^\psi]$, or string $s' = [a, \sigma_1^\psi, \sigma_0^\psi]$. Since both necessitated operations are done by the robot in all 3 options and (as per assumptions) the insertion of a is conflict-free to the operations σ_0^ψ and σ_1^ψ, the new string s' is still a valid solution to ψ. □

Theorem 3 *The addition of a character in a string that satisfies task ψ consisting of the OR operation only does not make the solution unsatisfiable.*

Proof Let the task be $\psi =$ "$\sigma_0^\psi \vee \sigma_1^\psi$" The smallest string that satisfies the task is either $s = [\sigma_0^\psi]$ or s $= [\sigma_1^\psi]$. Let a be added to the string at any general location, giving the modified string $s' = [\sigma_0^\psi, a]$, or string $s' = [a, \sigma_0^\psi]$ (and equivalent cases for $s = [\sigma_1^\psi]$). Since either of the necessitated operations are done by the robot in all the options and (as per assumptions) the insertion of a is conflict-free to the operations σ_0^ψ and σ_1^ψ, the new string s' is still a valid solution to ψ. □

Theorem 4 *The addition of a character in a string that satisfies task ψ consisting of sequence, AND, OR operations only does not make the solution unsatisfiable.*

Proof Let the task be ψ be recursively defined in terms of smaller sub-tasks ψ_1 and ψ_2 as (i) "$\psi = \psi_1$ THEN ψ_2", (ii) "$\psi = \psi_1 \wedge \psi_2$" or (iii) "$\psi = \psi_1 \vee \psi_2$". Let us assume that the theorem holds for the sub-tasks ψ_1 and ψ_2. Based on this assumption we prove that the theorem holds for all possible cases of breaking a larger task into smaller sub-tasks. Based on Theorems 1–3, the theorem holds for the 3 base cases of the smallest sub-tasks possible in a recursive definition. □

Case (i) Consider $\psi = \psi_1$ THEN ψ_2. Let s_1 be the string that satisfies ψ_1 and s_2 be the string that satisfies ψ_2, and s be a string formed by the concatenation of s_1 and s_2, $s = [s_1, s_2]$, that satisfies $\psi = \psi_1$ THEN ψ_2. If a character a is added in s_1, solution of the sub-task ψ_1, then the theorem holds as per the assumptions. If a character a is added in s_2, solution of the sub-task ψ_2, then the theorem holds as per the assumptions. If a character a is added in between the solutions, then $s' = [s_1, a, s_2]$ does everything specified in ψ_1, followed by everything specified in ψ_2, strictly performing ψ_1 before ψ_2. Therefore, s' is a valid solution to the task.

Case (ii) Consider $\psi = \psi_1 \wedge \psi_2$. Let s be a string that simultaneously satisfies both ψ_1 and ψ_2. Let the string s' be created by the addition of any character a to the string s. It is assumed that the theorem holds for the smaller sub-tasks. Therefore, s' satisfies ψ_1 and s' satisfies ψ_2. Since s' satisfies both ψ_1 and ψ_2, it satisfies $\psi = \psi_1 \wedge \psi_2$.

Case (iii) Consider $\psi = \psi_1 \vee \psi_2$. Let s be a string that satisfies either ψ_1 or ψ_2. Let the string s' be created by the addition of any character a to the string s. It is assumed that the theorem holds for the smaller sub-tasks. Therefore, if s satisfies ψ_1 then s' satisfies ψ_1. Similarly, if s satisfies ψ_2 then s' satisfies ψ_2. Either of the two cases shall hold since s satisfies ψ_1 or ψ_2, hence s' satisfies $\psi = \psi_1 \vee \psi_2$.

Theorem 5 *Consider a general string s which is a subsequence of a string P. If s satisfies the task ψ, then P also satisfies the task ψ. Let $s|P$ denote s as a subsequence of P. Hence Eq. (27) holds.*

$$s \vDash \psi \Rightarrow P \vDash \psi, \quad \text{if } s|P \tag{27}$$

Proof It is given that $s \models \psi$. From Theorem 4, the addition of any character a to s, giving a string s' also satisfies ψ. Therefore, let us add all characters that are in the super-sequence P but not in s one by one while maintaining the general ordering that they maintain in P. After all additions the string s would have transformed to P, while after every addition by Theorem 5, the validity of ψ would remain intact. □

Theorem 6 *Keeping the task trajectories $\left(\tau_\psi^{\text{task}}\right)$ used for the creation of a robot trajectory τ_r' fixed, the addition of a new character in the robot trajectory is wasteful. In other words, Eq. (28)*

$$\sigma \in \tau_r' \Rightarrow \left(\exists \psi \sigma \in \tau_\psi^{\text{task}}\right) \tag{28}$$

Proof Let $\tau_r' = \left[S_r, \sigma_1^r, \sigma_2^r, \ldots \sigma_i^r, \sigma_{i+1}^r, \ldots \sigma_{\text{len}(\tau_r')}^r\right]$, produced by keeping all the characters in the individual tasks assigned to the robot as per the super-sequence property. Let us further insert a new character 'a' in the string at any general position i, or $\tau_r'' = \left[S_r, \sigma_1^r, \sigma_2^r, \ldots \sigma_i^r, a, \sigma_{i+1}^r, \ldots \sigma_{\text{len}(\tau_r')}^r\right]$. Both the trajectories are valid for the task as per the super sequence property of Theorem 5. The costs of the two trajectories are given by Eqs. (29, 30)

$$\begin{aligned} C\left(\tau_r'\right) &= c\left(S_r, \sigma_1^r\right) + \sum_{j=1}^{\text{len}(\tau'(r))-1} c\left(\text{loc}\left(\sigma_j^r\right), \sigma_{j+1}^r\right) \\ &= c\left(S_r, \sigma_1^r\right) + \sum_{j=1}^{i-1} c\left(\text{loc}\left(\sigma_j^r\right), \sigma_{j+1}^r\right) \\ &\quad + c\left(\text{loc}\left(\sigma_i^r\right), \sigma_{i+1}^r\right) \\ &\quad + \sum_{j=i+1}^{\text{len}(\tau'(r))-1} c\left(\text{loc}\left(\sigma_j^r\right), \sigma_{j+1}^r\right) \end{aligned} \tag{29}$$

$$\begin{aligned} C\left(\tau_r''\right) &= c\left(S_r, \sigma_1^r\right) + \sum_{j=1}^{\text{len}(\tau''(r))-1} c\left(\text{loc}\left(\sigma_j^r\right), \sigma_{j+1}^r\right) \\ &= c\left(S_r, \sigma_1^r\right) + \sum_{j=1}^{i-1} c\left(\text{loc}\left(\sigma_j^r\right), \sigma_{j+1}^r\right) + c\left(\text{loc}\left(\sigma_i^r\right), a\right) + c\left(\text{loc}(a), \sigma_{i+1}^r\right) + \sum_{j=i+1}^{\text{len}(\tau'(r))-1} c\left(\text{loc}\left(\sigma_j^r\right), \sigma_{j+1}^r\right) \\ &= \left(c\left(S_r, \sigma_1^r\right) + \sum_{j=1}^{i-1} c\left(\text{loc}\left(\sigma_j^r\right), \sigma_{j+1}^r\right) + \sum_{j=i+1}^{\text{len}(\tau'(r))-1} c\left(\text{loc}\left(\sigma_j^r\right), \sigma_{j+1}^r\right)\right) + c\left(\text{loc}\left(\sigma_i^r\right), a\right) + c\left(\text{loc}(a), \sigma_{i+1}^r\right) \\ &= \left(C\left(\tau_r'\right) - c\left(\text{loc}\left(\sigma_i^r\right), \sigma_{i+1}^r\right)\right) + c\left(\text{loc}\left(\sigma_i^r\right), a\right) + c\left(\text{loc}(a), \sigma_{i+1}^r\right) \\ C\left(\tau_r''\right) &= C\left(\tau_r'\right) + c\left(\text{loc}\left(\sigma_i^r\right), a\right) + c\left(\text{loc}(a), \sigma_{i+1}^r\right) - c\left(\text{loc}\left(\sigma_i^r\right), \sigma_{i+1}^r\right) \end{aligned} \tag{30}$$

The operational cost of $c\left(\text{loc}\left(\sigma_i^r\right), a\right) + c\left(\text{loc}(a), \sigma_{i+1}^r\right)$ is larger than $c\left(\text{loc}\left(\sigma_i^r\right), \sigma_{i+1}^r\right)$, since the former performs an extra operation a. The navigation cost of $c\left(\text{loc}\left(\sigma_i^r\right), a\right) + c\left(\text{loc}(a), \sigma_{i+1}^r\right)$ is larger than $c\left(\text{loc}\left(\sigma_i^r\right), \sigma_{i+1}^r\right)$ due to the triangle law of inequality. In simpler terms, the triangle law of inequality holds for the cost function and hence we get Eq. (31).

$$\begin{aligned} &c\left(\text{loc}\left(\sigma_i^r\right), a\right) + c\left(\text{loc}(a), \sigma_{i+1}^r\right) \geq c\left(\text{loc}\left(\sigma_i^r\right), \sigma_{i+1}^r\right) \\ &c\left(\text{loc}\left(\sigma_i^r\right), a\right) + c\left(\text{loc}(a), \sigma_{i+1}^r\right) - c\left(\text{loc}\left(\sigma_i^r\right), \sigma_{i+1}^r\right) \geq 0 \end{aligned} \tag{31}$$

Using Eqs. (30) and (31) we get Eq. (32)

$$C\left(\tau_r''\right) \geq C\left(\tau_r'\right) \tag{32}$$

Hence, the cost of τ_r'' is larger than τ_r'. This makes it unnecessary to add elements to τ_r'. □

References

1. Baier C, Katoen JP (2008) Principles of model checking. MIT Press, Cambridge
2. Fisher M (2011) An introduction to practical formal methods using temporal logic. Wiley, West Sussex
3. Kress-Gazit H, Fainekos GE, Pappas GJ (2009) Temporal-logic-based reactive mission and motion planning. IEEE Trans Rob 25(6):1370–1381
4. Lahijanian M, Andersson SB, Belta C (2012) Temporal logic motion planning and control with probabilistic satisfaction guarantees. IEEE Trans Rob 28(2):396–409
5. Bhatia A, Kavraki LE, Vardi MY (2010a) Sampling-based motion planning with temporal goals. In: Proceedings of the 2010 IEEE international conference on robotics and automation, pp 2689–2696
6. Bhatia A, Kavraki LE, Vardi MY (2010b) Motion planning with hybrid dynamics and temporal goals. In: Proceedings of the 2010 49th IEEE conference on decision and control, pp 1108–1115
7. McMahon J, Plaku E (2014) Sampling-based tree search with discrete abstractions for motion planning with dynamics and temporal logic. In: Proceedings of the 2014 IEEE/RSJ international conference on intelligent robots and systems, pp 3726–3733
8. Svorenova M, Tumova J, Barnat J, Cerna I (2012) Attraction-based receding horizon path planning with temporal logic constraints. In: Proceedings of the 2012 IEEE 51st annual conference on decision and control, pp 6749–6754
9. Lahijanian M, Almagor S, Fried D, Kavraki LE, Vardi MY (2015) This time the robot settles for a cost: a quantitative approach to temporal logic planning with partial satisfaction. In: Proceedings of the twenty-ninth AAAI conference on artificial intelligence. AAAI, pp 3664–3671
10. Smith SL, Tumova J, Belta C, Rus D (2011) Optimal path planning for surveillance with temporal-logic constraints. Int J Robot Res 30(14):1695–1708
11. Svorenova M, Cerna I, Belta C (2015) Optimal temporal logic control for deterministic transition systems with probabilistic penalties. IEEE Trans Autom Control 60(6):1528–1541
12. Ulusoy A, Smith SL, Ding XC, Belta C, Rus D (2013) Optimality and robustness in multi-robot path planning with temporal logic constraints. Int J Robot Res 32(8):889–911
13. Fu J, Atanasov N, Topcu U, Pappas GJ (2016) Optimal temporal logic planning in probabilistic semantic maps. In: Proceedings of the 2016 IEEE international conference on robotics and automation, Stockholm, pp 3690–3697
14. Lacerda B, Parker D, Hawes N (2014) Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications. In: Proceedings of the 2014 IEEE/RSJ international conference on intelligent robots and systems, pp 1511–1516
15. Schillinger P, Bürger M, Dimarogonas DV (2018) Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. Int J Robot Res 37(7):818–838
16. Faruq F, Parker D, Laccrda B, Hawes N (2018) Simultaneous task allocation and planning under uncertainty. In: Proceedings of the 2018 IEEE/RSJ international conference on intelligent robots and systems, pp 3559–3564
17. Torres J, Baier JA (2015) Polynomial-time reformulations of LTL temporally extended goals into final-state goals. In: Proceedings of the international joint conference on artificial intelligence, pp 1696–1703
18. Camacho A, Triantafillou E, Muise C, Baier J, McIlraith S (2017) Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In: Proceedings of the AAAI conference on artificial intelligence, vol 31, No. (1)
19. Menghi C, Garcia S, Pelliccione P, Tumova J (2018) Multi-robot LTL planning under uncertainty. In: Havelund K, Peleska J, Roscoe B, de Vink E (eds) Formal methods. Lecture notes in computer science, vol 10951. Springer, Cham, pp 399–417
20. Karaman S, Frazzoli E (2009) Sampling-based motion planning with deterministic μ-calculus specifications. In: Proceedings of the 48h IEEE conference on decision and control (CDC) held jointly with 2009 28th Chinese control conference, Shanghai, pp 2222–2229
21. Kuffner JJ, LaValle SM (2000) RRT-connect: an efficient approach to single-query path planning. In: Proceedings IEEE international conference on robotics and automation, pp 995–1001
22. LaValle SM, Kuffner JJ (1999) Randomized kinodynamic planning. In: Proceedings of the IEEE international conference on robotics and automation, pp 473—479
23. Kantaros Y, Zavlanos MM (2018) Temporal logic optimal control for large-scale multi-robot systems: 10400 states and beyond. In: Proceedings of the 2018 IEEE conference on decision and control, Miami Beach, FL, pp 2519–2524
24. Kantaros Y, Zavlanosm MM (2019) Sampling-based optimal control synthesis for multirobot systems under global temporal tasks. IEEE Trans Autom Control 64(5):1916–1931
25. Xidias EK, Azariadis PN (2011) Mission design for a group of autonomous guided vehicles. Robot Auton Syst 59(1):34–43
26. Lu L-C, Yue T-W (2019) Mission-oriented ant-team ACO for min–max MTSP. Appl Soft Comput 76:436–444
27. Kala R (2016) Sampling based mission planning for multiple robots. In: Proceedings of the IEEE congress on evolutionary computation, pp 662–669
28. Şenol MB (2019) A mixed integer programming (MIP) model for evaluating navigation and task planning of human–robot interactions (HRI). Intel Serv Robot 12:231–242
29. Kala R, Khan A, Diksha D, Shelly S, Sinha S (2018) Evolutionary mission planning. In: Proceedings of the 2018 IEEE congress on evolutionary computation, pp 1–8
30. Edelkamp S, Jabbar S, Nazih M (2006) Large-scale optimal PDDL3 planning with MIPS-XXL. In: 5th international planning competition booklet
31. Gerevini A, Haslum P, Long D, Saetti A, Dimopoulos Y (2009) Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. Artif Intell 173(5–6):619–668
32. Cambon S, Alami R, Gravot F (2009) A hybrid approach to intricate motion, manipulation and task planning. Int J Robot Res 28(1):104–126
33. Galindo C, Fernandez-Madrigal J, Gonzalez J (2008) Multihierarchical interactive task planning: application to mobile robotics. IEEE Trans Syst Man Cybern Part B Cybern 38(3):785–798
34. Srivastava S, Fang E, Riano L, Chitnis R, Russell S, Abbeel P (2014) Combined task and motion planning through an extensible planner-independent interface layer. In: Proceedings of the 2014 IEEE international conference on robotics and automation, Hong Kong, pp 639–646
35. Matoui F, Boussaid B, Metoui B et al (2020) Contribution to the path planning of a multi-robot system: centralized architecture. Intel Serv Robot 13:147–158
36. Kala R (2018) On repelling robotic trajectories: coordination in navigation of multiple mobile robots. Intel Serv Robot 11:79–95
37. Lagoudakis MG, Berhault M, Koenig S, Keskinocak P, Kleywegt AJ (2004) Simple auctions with performance guarantees for multi-robot task allocation. In: Proceedings of the 2004 IEEE/RSJ international conference on intelligent robots and systems, pp 698–705
38. Choset H, Lynch KM, Hutchinson S, Kantor GA, Burgard W, Kavraki LE, Thrun S (2005) Principles of robot motion: theory, algorithms, and implementations. MIT Press, Cambridge

39. Tiwari R, Shukla A, Kala R (2013) Intelligent planning for mobile robotics: algorithmic approaches. IGI Global Publishers, Hershey
40. Kavraki LE, Svestka P, Latombe JC, Overmars H (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans Robot Autom 12(4):566–580
41. Russell S, Norvig P (2010) Problem solving by searching. In: Artificial intelligence: a modern approach. Pearson, Upper Saddle River, pp 64–119