Task Assignment for Complex UAV Operations using Genetic Algorithms

Sertac Karaman *   Tal Shima †   Emilio Frazzoli *

Process algebra has been recently proposed as a tool to specify complex missions, e.g., for UAV applications. In this paper, we present a novel genetic algorithm that is applicable to mission planning and optimization, with process algebra specifications. An example is provided with reference to UAV base defence assignment operations. Solutions to the assignment problem are given in the form of chromosomes that are manipulated by evolutionary based operators to provide better solutions in the next generation. The crossover and mutation operators are formally defined using the process algebra methodology, along with specific algorithms needed for their execution. The viability of the approach is investigated using simulations of small and large scale problems. It is shown that, for a small sized problem, the algorithm converges to the optimal solution.

I. Introduction

Uninhabited aerial vehicles (UAVs) are becoming increasingly effective in performing missions that have previously been performed by manned airplanes. Their efficiency mainly stems from the lack of an on-board human operator. This enables developing systems with significant weight savings, lower costs, and that can perform long endurance tasks. Currently, basic tasks of UAVs such as flying and trajectory planning from way-point to way-point can be automatized. To enable the simultaneous cooperative operation of multiple such systems in complex missions, higher levels of autonomy are constantly sought. Within the last decade, cooperative control algorithms have been proposed to preferably optimally coordinate multiple UAVs for such missions (see for example Refs. 1–4).

Recently, cooperative control algorithms have been extended to handle more complex tasks and constraints, called the mission specifications, which are expressed in a formal and natural manner (see for example Refs. 5,6). These mission specifications include, but are not limited to, combinations of temporal ordering among tasks as well as conjunctive and disjunctive logical constraints. More recent work has also considered specification languages with strict deadlines to specify and solve more complex UAV missions (see for example Ref. 7).

In Ref. 6 Process Algebra (PA) is used to specify a class of complex coupling constraints between tasks in UAV missions. This paper also adopts PA as the specification language for reasons to be outlined shortly. In computer science, process algebra is used for reasoning about the time behavior of software. For such reasoning, the software is assumed to be able to execute actions from a set A in some order. An action is a very general abstraction; it may refer to a blink of light, writing into a file, or moving a robotic arm. A behavior of the system is, then, a sequence of actions which are ordered with respect to the order that they were executed by the software. The sequence \((a_1, a_2, \ldots, a_n)\) for which \(a_1, a_2, \ldots, a_n \in A\), for instance, would be a behavior of the system. Then, using the process algebra terms one can indeed specify the set of behaviors that a system can exhibit. This can be used as a design specification for automatic generation of software; or it can be used for checking whether a given software satisfies such criteria.

In most military multiple-UAV missions, individual tasks like area search, classifying, or destroying a target, can be coupled with each other with temporal and logical constraints. Intuitively, these tasks, which we will refer to as atomic objectives, correspond to the actions, whereas the coupling constraints will be represented by the process algebra terms. More precisely, tying the atomic objectives together using process algebra terms, high level tasks will be described. We will denote such high level tasks as complex objectives. In the end, the entire UAV mission will be given as a single specification, i.e., a single complex objective, using which the optimal plan that satisfies this specification is generated automatically.

The assignment of multiple cooperating UAVs to multiple tasks, such as collections of the atomic objectives mentioned above, requires the solution of a combinatorial optimization problem. Genetic algorithms (GAs) are a well known stochastic search method that can be used to solve such problems. GAs involve iteratively manipulating a population of candidate solutions, encoded in chromosomes, to obtain a population that includes better solutions. The algorithm consists of the following basic steps: 1) Generation of an initial population, 2) Evaluation of the fitness of each chromosome in the population, 3) Stopping if an end condition is satisfied and continuing if not, 4) Creating new candidate chromosomes by applying genetic operators (commonly selection, crossover, mutation, and elitism), thus

---

*Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 02139; {sertac, frazzoli}@mit.edu
†Faculty of Aerospace Engineering, Technion - Israel Institute of Technology, Haifa, 32000, Israel; tal.shima@technion.ac.il  Associate Fellow AIAA
simulating nature’s evolution process, 5) replacement of old chromosomes by new ones, 6) looping back to step 2. The encoding of the GA chromosome and definition of the genetic operators mentioned above are the major part of the solution process and are problem specific.

Our work in this paper is mostly related to our previous work in Refs. 3–5. In Ref. 5,7,8, formal languages such as Linear Temporal Logic and Metric Temporal Logic were employed to describe complex tasks and constraints in military UAV operations. To handle larger-scale problems more effectively, in Ref. 6, computationally more tractable process algebra specifications were incorporated into a tree search based task assignment algorithm (see Ref. 3 for the details of tree search). Process algebra type specifications, if not as expressive as the temporal logics, were shown to describe a broad class of complex mission specifications of practical importance in Ref. 6. Genetic algorithms, on the other hand, were used in Refs. 9,10 to improve the computational effectiveness of the task assignment algorithms, however, their integration with complex mission specifications were never considered. This paper fills in this gap by proposing a computationally effective algorithm, which can yet handle a broad class of complex tasks specified using process algebra.

The main contribution of this paper is a genetic algorithm solution to a multiple-UAV mission planning problem with complex mission specifications represented via process algebra terms. The paper is organized as follows. In Section II., related work on applications of genetic algorithms to multiple-UAV task assignment problems is outlined and in Section III. the process algebra specification framework is introduced. Section IV. presents the specification of complex multiple UAV missions using PA. Then, in Section V. the GA-based task assignment algorithm that can handle PA specifications is given followed by a simulation study presented in Section VI.. Finally, the paper is concluded in Section VII..

II. UAV Task Assignment using Genetic Algorithms

Many combinatorial optimization problems deal with the assignment of an agent or a group of agents to a set of tasks. This set may define a single job, problem, or mission. The term agent can relate to a human or machine performing the tasks. An agent has specific capabilities, incurs a cost for fulfilling a task, and has a limited budget that can not be exceeded. The objective is commonly to find a feasible assignment that minimizes some cost function. In UAV operations this cost can relate for example to the time needed or distance traveled to fulfill the set of tasks.

One of the most fundamental combinatorial optimization problems is the travelling salesman problem (TSP). In such a problem the goal is to find the minimal distance tour of a salesman visiting a set of cities. The cost incurred is the distance traveled between each set of cities. The agent must visit each city once and the tour must be closed, i.e. the agent must return to its initial location (depot). TSP can be modelled as a graph with vertices corresponding to cities and edges corresponding to connections between cities with their length equal to the distance between the two connected cities.

The TSP can be generalized to include multiple agents. In such a problem multiple agents are dispatched from a single depot to visit the set of cities. The task is to find the best (minimal cost) closed tours for the agents such that each city in the set is visited by a single agent. By incorporating some additional constraints this problem can be extended to a wide variety of vehicle routing problems (VRP). In such problems multiple vehicles, with a given capacity, are dispatched from a single depot to deliver to n customers each requiring a specified weight of goods, and then return to the depot. Extensions to these problems may include multiple depots, time windows for visiting specific locations, etc. The performance in such problems involving multiple agents is strongly dependent on cooperation between the agents. The outcome obtained by properly implemented cooperation has synergistic effects at the total task execution, i.e. the performance of the total assignment will be better than one reached by a group of individual agents working separately in the same space on a single or series of tasks. Such is the case with the problem of assigning multiple UAVs to perform tasks on multiple targets.

The significant difficulty in solving many combinatorial optimization problems is that they are NP-hard and therefore could not be solved in polynomial time by deterministic methods. So, due to the prohibitive computational complexity of the problem, the traditional deterministic search algorithms provide an optimal solution only for small-sized problems. For large sized problems they may provide a feasible solution in a given bounded run-time. Approximate algorithms can also be used for solving such problems. These algorithms give a solution of cost J to the problem with optimal cost J* such that the ratio J/J* is bounded. Many approximate algorithms exist for solving TSP including the ones presented in Ref. 14.

If optimality is not sought and the goal is to obtain fast a good feasible solution, then stochastic search algorithms that employ a degree of randomness as part of their logic can be used. An algorithm of this type uses a random input to guide its behavior in the hope of achieving good performance in the “average case” and converge to a good solution in the expected runtime. Evolutionary algorithms (EA), which are inspired by the mutation selection process witnessed in nature, are a common stochastic search method used to solve many combinatorial optimization problems. These methods involve iteratively manipulating populations of solutions, termed chromosomes, that encode candidate solutions. The generational process is performed by applying evolutionary operators like selection, crossover, and mutation. Candidate solution selection is performed by evaluating the fitness of each chromosome in the population.
Historically, there were three main types of EAs: GA, evolutionary strategies, and evolutionary programming, with GA being the most popular one.\textsuperscript{15}

Much work in applying GAs to the TSP\textsuperscript{16} is concerned with the encoding of the chromosomes and the use of special crossover operators that preserve its validity. In Ref. 17 the VRP was solved using a pure GA and a hybrid of the GA with neighboring search methods. Improved performance with respect to solution time and quality was shown compared to other stochastic search methods such as simulated annealing and Tabu search. Recently, GA have been proposed for solving UAV cooperative assignment problems\textsuperscript{9,10}. In Ref. 9 a GA was proposed for a scenario where multiple homogeneous UAVs cooperate in performing multiple tasks (such as classify, attack, and verify) on multiple stationary ground targets. Solving such problems required assigning different tasks to different vehicles and consequently assigning each vehicle with a flyable path that it must follow. In Ref. 10 GA was used to solve a cooperative UAV assignment problem where targets required simultaneous actions from several UAVs. In both these studies simulation results showed the effectiveness of GAs in providing in real-time good feasible solutions to these problems.

III. Process Algebra

Most engineering systems have a set of actions that allows it to communicate with the outer world or manipulate the objects therein to accomplish a high-level task. Of course, in this context, the definition of the high-level task, as well as the actions, depend on the granularity of the abstraction; but we will assume that these notions are such that the system under consideration will be designed to handle only one high-level task and the actions are atomic in the sense that they can not be accomplished by executing a sequence of other actions.

Even though systems that do not terminate and operate in a persistent manner exist and they are interesting in their own right, most of the real-world systems execute a sequence of actions, which eventually lead to accomplishment of the high-level task and termination of the system. In the rest of the paper, we will assume that each high-level task, if it can be accomplished at all, can be accomplished by a terminating execution of the system, i.e., a finite sequence of actions. Such an execution is called a “behavior” of the system. It is important to note at this point that, most of the time, such a behavior that leads to successful fulfillment of the requirements is not unique and it is crucial and challenging to naturally and formally “specify” the set of desired behavior of a system. Moreover, given the specification, designing algorithms that automatically design the system to fulfill the specification is important and challenging in its own right.

Along with many other formalizations such as temporal logics\textsuperscript{18}, the $\mu$-calculus\textsuperscript{19}, or the Petri nets\textsuperscript{20}, process algebra\textsuperscript{21–23} is a methodology that can be used to specify the desired behavior of a system. Initiated and used in Computer Science to reason about computer software, process algebras found many applications in several diverse fields from web applications to hybrid systems (see for example\textsuperscript{24}). Using process algebra for specification of UAV missions was first considered in Ref. 6. This section presents an introduction to the process algebra based specification framework of Ref. 6 followed by scenarios of interest for further clarification.

Before going into formal definitions and statements, let us introduce the following notation. A finite sequence (of distinct elements) on a set $S$ is an injective map $\sigma$ from $\{1, 2, \ldots, K\}$ to $S$ where $K$ is a finite number. A finite sequence will often be written as $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(K))$. An element $s$ is said to be an element of a finite sequence $\sigma$, denoted by $s \in \sigma$ with a slight abuse of notation, if there exists $k \in \{1, 2, \ldots, K\}$ such that $\sigma(k) = s$.

The notation $|\sigma|$ is used to denote the number of elements of a sequence $\sigma$, i.e., $|\sigma| = K$. Given two sequences $\sigma_1$ and $\sigma_2$ both defined on a same set $S$, we will denote their concatenation by $\sigma_1[\sigma_2]$, which itself is also a finite sequence defined on the set $S$ with domain $\{1, 2, \ldots, |\sigma_1| + |\sigma_2|\}$. More precisely, $(\sigma_1[\sigma_2](k) = \sigma_1(k)$ for all $k \in \{1, 2, \ldots, |\sigma_1|\}$ and $(\sigma_1[\sigma_2](|\sigma_1| + k) = \sigma_2(k)$ for all $k \in \{1, 2, \ldots, |\sigma_2|\}$. For any two elements $s_1, s_2 \in S$, the ordering relation $<_\sigma$ defined on the elements of the sequence $\sigma$ is formalized as follows: $s_1 <_\sigma s_2$ if there exists $i, j \in \{1, 2, \ldots, K\}$ with $i < j$ such that $\sigma(i) = s_1$ and $\sigma(j) = s_2$. Given a sequence $\sigma$ defined on a set $S$, an order preserving projection of $\sigma$ on to a set $S' \subseteq S$ is defined as the sequence $\sigma'$, for which the following hold: (i) for all $s \in \sigma$, we have that if $s \in S'$ then $s \in \sigma'$ holds, i.e., $\forall s \in \sigma, (s \in S' \Rightarrow s \in \sigma')$; (ii) for any $s_1, s_2 \in \sigma'$, if $s_1 <_\sigma s_2$ holds, then $s_1 <_\sigma s_2$ must hold, i.e., $\forall s_1, s_2 \in \sigma', (s_1 <_\sigma s_2 \Rightarrow s_1 <_\sigma s_2)$. Given a sequence $\sigma$ defined on a set $S$, its order preserving projection on to a set $S' \subseteq S$ will be denoted as $[\sigma]_{S'}$. For the sake of brevity, a finite sequence will be simply referred to as a sequence from this point on and given a set $S$, we denote the set of all sequences on set $S$ by $\Sigma_S$.

An important notion in process algebra is the definition of the set of terms, which is formally given as follows:

\begin{definition}[Terms of Process Algebra] Given a finite set $A$ of actions, the set $T$ of PA terms is defined inductively as follows:
\begin{itemize}
  \item each action $a \in A$ is in $T$;
  \item if $p, p' \in T$, then $p + p' \in T$, $p \cdot p' \in T$, and $p \parallel p' \in T$.
\end{itemize}
\end{definition}
Terms are related to each other as they can evolve from one to another. This evolution is made through a transition, denoted as $p \xrightarrow{a} p'$, where $p, p' \in \mathbb{T}$, and $a \in A$. This transition is read as “Process $p$ can evolve into process $p'$ by executing the action $a$.” There is also a special process, denoted as $\checkmark$, which corresponds to the terminated process. By definition, the process $\checkmark$ has no actions to execute and can not evolve into any other process.

Following the definition of the set of terms, each action can be a term such as $p = a$, where $a \in A$. Informally speaking, the system specified as $a$ has only one behavior: it can execute $a$ and then terminate, which is denoted as $a \xrightarrow{a} \checkmark$. To specify more complex systems, the PA specification framework offers the operators $\{+, (\cdot), (\parallel)\}$, which are called the alternative, sequential, and parallel composition operators, respectively. Intuitively, a process that is specified with the term $p + p'$ behaves either like $p$ or $p'$, i.e., either executes a behavior of $p$ or executes one of $p'$ (but not both). The term $p \cdot p'$, on the other hand, first executes a behavior of the process $p$, and right after $p$ terminates, it executes a behavior of $p'$. The process $p \parallel p'$ is said to terminate when $p'$ terminates. The process $p \parallel p'$ executes a behavior of each of $p$ and $p'$ concurrently. The process $p \parallel p'$ is said to have terminated, when both $p$ and $p'$ terminate.

This informal presentation of the behavior of processes can be formalized by the operational semantics, which is defined as a set of transition system specifications (TSSs). A TSS is composed of a set $H$ of premises and a conclusion $\pi$, denoted as $H \vdash \pi$, where $\pi$ is a transition and $H$ is a set of transitions. A TSS states that if the premises $H$ are possible transitions, then so is the transition $\pi$. The semantics (meaning) of each PA term is defined using the operational semantics of process algebra given as follows:

**Definition III.2 (Operational Semantics of PA)** The operational semantics of the process algebra is given by the following set of transition system specifications:

$$
\begin{align*}
\frac{p_1 \xrightarrow{a} \checkmark}{p_1 + p_2 \xrightarrow{a} \checkmark} & \quad \frac{p_1 \xrightarrow{a} p_1'}{p_1 + p_2 \xrightarrow{a} p_1'} & \quad \frac{p_2 \xrightarrow{a} \checkmark}{p_1 + p_2 \xrightarrow{a} \checkmark} \\
\frac{p_1 \xrightarrow{a} \checkmark}{p_1 \parallel p_2 \xrightarrow{a} \checkmark} & \quad \frac{p_1 \xrightarrow{a} p_1'}{p_1 \parallel p_2 \xrightarrow{a} p_1'} & \quad \frac{p_2 \xrightarrow{a} \checkmark}{p_1 \parallel p_2 \xrightarrow{a} \checkmark} & \quad \frac{p_2 \xrightarrow{a} p_2'}{p_1 \parallel p_2 \xrightarrow{a} p_2'}
\end{align*}
$$

where $a \in A$ and $p_1, p_1', p_2, p_2' \in \mathbb{T}$

Notice that the first TSS formally states that any action $a \in A$ can execute $a$ and then evolve to the terminating process. Similarly, the other TSSs specify the semantics of the alternative, sequential, and parallel composition operators.

With its recursive definition, the operational semantics associates each process with a set of traces that the process can execute. This set is merely the set of all behaviors of the system. More formally, any sequence $\sigma = (a_1, a_2, \ldots, a_k)$ of actions is called a trace of a term $p_0$ if and only if there exists a set of processes $p_1, p_2, \ldots, p_k$ such that $p_{i-1} \xrightarrow{a_i} p_i$ for all $i \in \{1, 2, \ldots, k\}$ and $p_k = \checkmark$. In other words, a trace of a process is a behavior of the process as it is a sequence of its available actions which lead to successful termination. Moreover, the set of all such traces of a process $p_0$, which will be denoted by $\Gamma_{p_0}$, formally defines the set of behavior of the process $p_0$.

Each process algebra term can be represented by a special data structure called a parse tree. The parse tree of a process algebra term is composed of nodes, each of which encodes either an operator or an atomic objective. More precisely, each leaf node in the tree encodes an atomic objective and every other node (this includes the root if the parse tree is not a single node) encodes an operator.

Given a term $p \in \mathbb{T}$ its parse tree is recursively defined as follows:

- If $p \in A$, i.e., $p$ is an action itself, then the parse tree of $p$ is a single node which is labeled with $p$.
- If $p = p_1 \bowtie p_2$, where $\bowtie \in \{, +, \parallel\}$, then the parse tree of $p$ is a tree which is rooted at a node labeled with $\bowtie$ and has the parse tree of $p_1$ and $p_2$ as its left and right children, respectively.

Some of the algorithms that will be introduced in the next sections will run on the parse trees of the given specification. To render these algorithms more readable, let us present some notation, which will be used throughout the paper. Let $N_p$ denote the set of nodes in the parse tree of a process $p$ and let $n$ be a node from the set $N_p$. Then, the function $Parent(n) : N_p \rightarrow N_p \cup \{\delta\}$ returns the parent node of a given node. If $n$ has no parent, then we have $Parent(n) = \delta$. The function $Leaf(n) : N_p \rightarrow \{false, true\}$ returns true if node $n$ is a leaf, i.e., it has no children, and false otherwise. As mentioned before, each node in the tree encodes either an operator or an atomic objective. The functions $Operator(n) : N_p \rightarrow \{+, \cdot, \parallel\}$ and $AtomicObjective(n) : N_p \rightarrow O$ return the encoded operator and...
the atomic objective, respectively. While the function \( \text{AtomicObjective} \) is defined only for the leaf nodes, \( \text{Operator} \) is defined for all the other nodes in \( N_p \). Finally, the function \( \text{Children}(n) : N_p \rightarrow \Sigma_{N_p} \) maps each node to an ordered sequence of its children such that the left child is the first element of the sequence, whereas the right child is the second one.

### IV. Specification of Complex Multiple-UAV Missions

In this section, the process algebra framework is employed to specify a class of multiple-UAV missions. Even though most of the definitions of this section can be found in Ref. 6, we provide them for the purpose of completeness. The reader is referred to Ref. 6 for a more thorough discussion of the material to follow.

#### A. Atomic Objectives

The UAV missions discussed in this paper consist of a set \( U = \{ u_1, u_2, \ldots, u_N \} \) of \( N \) UAVs and a set \( \mathcal{O} \) of atomic objectives, which are those tasks in the mission that can not represented by a combination of any others. In essence, atomic objectives are abstractions of individual tasks in a UAV mission. To the authors knowledge, the first such abstractions were presented recently in Ref. 25 and further developed and employed in Ref. 6.

**Definition IV.1** An atomic objective \( o \) is a tuple \( (x_o^1, x_o^2, T_o^e, U_o) \) where

- \( x_o^1 \in \mathbb{R}^2 \) is the entry point
- \( x_o^2 \in \mathbb{R}^2 \) is the exit point
- \( T_o^e \in \mathbb{R}_{\geq 0} \) is the execution time
- \( U_o \in U \) is a capable UAV

Intuitively, only \( U_o \) can execute the atomic objective \( o \) in exactly \( T_o^e \) amount of time. UAV \( U_o \) moves to the coordinate \( x_o^1 \) to start executing \( o \) and ends up at coordinate \( x_o^2 \) after the execution.

We will assume without any loss of generality that \( o \) can not be executed by any UAV other than \( U_o \). Later in the paper, we will show that the tasks that can be executed by one of several UAVs can be represented as a combination of different atomic objectives.

The definition of atomic objectives are quite general and capture many different types of individual tasks, including sector or area search, classification, attack, rescue, target tracking, reconnaissance, et cetera\(^{25} \). Recall that our main motivation is to be able to represent the mission specification, i.e., the coupling constraints between the atomic objectives, in a formal and natural manner; this idea is formalized next.

#### B. Schedules, Observations, and Specifications

A single UAV schedule is a sequence of atomic objective and time pairs. Intuitively, a single UAV schedule \( \sigma_u \) associates with the UAV \( u \in U \), a list of atomic objectives to be executed along with their execution times. More precisely, if \( (o, t) \in \sigma_u \) for some \( o \in \mathcal{O} \) and \( t \in \mathbb{R}_{\geq 0} \), then the atomic objective \( o \) is said to be scheduled to be executed at time \( t \) by UAV \( u \). A complete schedule, then, is a set \( \mathcal{P} \) of single UAV schedules such that \( \mathcal{P} \) contains exactly one single UAV schedule for each UAV in \( U \).

Given two atomic objectives \( o_i \) and \( o_j \), let us denote the time it takes for UAV \( u \) to travel from the initial coordinate of \( o_i \) to the final coordinate of \( o_j \) as \( T_{o_i-o_j}^u \). Then, a complete schedule \( \mathcal{P} \) is said to be valid if for each UAV \( u \in U \) and for all pairs \( (o_i, t_i) \in \sigma_u \) the atomic objective \( o_i \) can indeed be executed at time \( t_i \) by UAV \( u \). More precisely, for any \( \sigma_u = \{(o_1,t_1),(o_2,t_2),\ldots,(o_k,t_k)\} \) in \( \mathcal{P} \), the following holds: \( t_{i-1} + T_{o_{i-1}o_i}^u \leq t_i \) for all \( i \in \{2, \ldots, k\} \).

A sequence \( \pi = (o_1, o_2, \ldots, o_k) \) of atomic objectives is called an observation of the schedule \( \mathcal{P} \) if (i) any atomic objective \( o \) that is scheduled in \( \mathcal{P} \) is an element of \( \pi \) and (ii) for each atomic objective \( o \in \pi \) there exists a real number \( t \) such that \( t \leq t_i \leq t + T_o^e \), where \( t_i \) is such that \( (o, t) \in \sigma_u \) for some \( u \in U \), \( T_o^e \) is the execution time of \( o \), and \( o_i \leq \pi o_j \) if and only if \( t_i \leq t_j \).

Intuitively, an observation is a sequence \( \pi \) of atomic objectives such that corresponding to each \( o_i \) that appear in \( \pi \) one can find a time instance \( t_i \) within the execution interval of the atomic objective \( o_i \) so that the ordering of these time instances are the same as the ordering of their corresponding atomic objectives in \( \pi \). From here on, we will denote the set of all observations of a valid complete schedule \( \mathcal{P} \) by \( \Pi_\mathcal{P} \).

Following the definition of observations, a specification and its satisfaction is formalized as follows.

**Definition IV.2 (Specification)** A specification is a process algebra term defined on the set \( \mathcal{O} \) of atomic objectives. A schedule \( \mathcal{P} \) is said to satisfy a specification \( p \) if and only if any observation of \( \mathcal{P} \) is a trace of \( p \), i.e., \( \Pi_\mathcal{P} \subseteq \Gamma_p \) holds.

These preliminary definitions will be used to formalize the problem definition in the next section.
Given a schedule \( \mathcal{P} = \{ \sigma_u \mid u \in \mathcal{U} \} \), each single UAV schedule \( \sigma_u \) in \( \mathcal{P} \) can be naturally associated with a real number \( \tau_u \), which represents the time that UAV \( u \) is finished with the execution of its last atomic objective. More formally, \( \tau_u = t' + T_{o'} \), where \( (o', t') = \sigma_u(\{\sigma_u\}) \). The real number \( \tau_u \) will be referred to as the completion time of \( \sigma_u \). Using the completion times \( \{\tau_u\}_{u \in \mathcal{U}} \), it is possible to define the cost of the schedule \( \mathcal{P} \) in many ways. Two of the common cost functions include the total completion time \( J_1(\mathcal{P}) \) and maximum completion time \( J_2(\mathcal{P}) \), which are defined, respectively, as

\[
J_1(\mathcal{P}) = \sum_{u \in \mathcal{U}} \tau_u, \quad J_2(\mathcal{P}) = \max_{u \in \mathcal{U}} \tau_u.
\]

Then, the problem we focus on in this paper can be formalized as follows.

**Problem IV.3** Given a set \( \mathcal{U} \) of UAVs, a set \( \mathcal{O} \) of atomic objectives, traveling times \( T_{o,u,o_j}^i \) for all \( u \in \mathcal{U} \) and all \( o_i, o_j \in \mathcal{O} \), and a process algebra specification \( p_{\text{spec}} \) defined on \( \mathcal{O} \), the Optimal Planning Problem for Complex Multiple-UAV Missions is to find a valid schedule \( \mathcal{P} \) which minimizes the cost metric \( J_1(\mathcal{P}) \) (or \( J_2(\mathcal{P}) \)) and satisfies the specification \( p_{\text{spec}} \).

Recently, a tree search based solution to Problem IV.3 was given in Ref. 6, which extends the previous recent results by Ref. 3 to handle process algebra specifications. The tree search algorithm presented in these references effectively searches the state space of all solutions and returns a feasible solution to the problem in time polynomial with respect to the size of the specification \( p_{\text{spec}} \) as well as the number of UAVs. Moreover, given time the algorithm improves the existing solution with the guarantee of termination with an optimum solution in finite time. In the next section, we provide a Genetic Algorithm heuristic solution to Problem IV.3.

### V. Process Algebra based Genetic Algorithm

In this section, we introduce an effective genetic algorithm search strategy for the task assignment problem of complex UAV operations given in problem IV.3. Given a specification \( p \), a chromosome \( X \) is formally defined as a single trace of \( p \), i.e., \( X \in \Gamma_p \). The GA maintains a set of chromosomes \( X \) called the generation. Before starting the algorithm an initial generation is created by constructing several random chromosomes. Then, at each step, (i) parent chromosomes are selected from \( X \) according to their fitness, (ii) new chromosomes are generated from the parents using the crossover operation, (iii) some of the chromosomes are mutated randomly, and (iv) the chromosomes that are more fit than others are carried to the next generation. In what follows, we detail these four operations, after presenting the relationship between the schedules and the chromosomes. Near the end of this section, we present the GA as a whole and point out some of its interesting properties.

#### A. Evolutionary Operators

Each chromosome \( X_i \) (hence every trace of the specification \( p \)) naturally corresponds to a valid complete schedule \( \mathcal{P}(X_i) \). Before formalizing the construction of \( \mathcal{P}(X_i) \), let us introduce the following definition. An atomic objective \( \bar{o} \) is said to be a predecessor of another atomic objective \( o \) in a specification \( p \) if the following two conditions hold: (i) there exists a trace \( \gamma \) of \( p \), in which \( \bar{o} \) appears before \( o \) in \( \gamma \), i.e., \( \bar{o} <_{\gamma} o \), (ii) there is no trace of \( p \), in which \( o \) appears before \( \bar{o} \). Equivalently, it can be shown that \( \bar{o} \) is a predecessor of \( o \) in \( p \) if and only if the parse tree of \( p \) includes a node that binds two terms \( p_1 \) and \( p_2 \) with a sequential composition operator as in \( p_1 \cdot p_2 \), where \( p_1 \) includes \( o_1 \) and \( p_2 \) includes \( o_2 \) as an atomic objective (see Ref. 6). We will denote the set of all predecessors of a given atomic objective \( o \) in a specification \( p \) by \( \text{Pred}_p(o) \). Notice that the sets \( \text{Pred}_p(o) \) for all \( o \in \mathcal{O} \) can be formed efficiently by observing the parse tree of \( p \). This process can be executed only once before starting the algorithm in time that is bounded by a polynomial in the size of \( p \).

Given the definition of predecessors, the complete schedule \( \mathcal{P}(X_i) \) is generated as follows. A chromosome with only one atomic objective, i.e., \( X = (\bar{o}) \) with \( \bar{o} = (x_1^o, x_2^o, T_o, u_i) \), corresponds to the schedule \( \mathcal{P} = \{ \sigma_{u_1}, \sigma_{u_2}, \ldots, \sigma_{u_N} \} \), where \( \sigma_{u_i} = (\bar{o}, t) \), with \( t \) being the time it takes \( u_i \) to travel from its initial position to \( x_1^o \), and for all \( u_j \neq u_i \) and \( u_j \in \mathcal{U} \), \( u_j \) is an empty sequence. Given a chromosome \( X \) with \( |X| = K > 1 \), let \( o = (x_1^o, x_2^o, T_o, u_i) \) be the last atomic objective that appears in \( p \), i.e., \( X = X'_{(\bar{o})} \), \( |X'| = K - 1 \) and \( X'(k) = X(k) \) for all \( k \in \{1, 2, \ldots, K - 1\} \). Let us also denote \( \mathcal{P}(X') \) as \( \mathcal{P}(X') = \{ \sigma_{u_1}', \sigma_{u_2}', \ldots, \sigma_{u_N}' \} \). Then, \( \mathcal{P}(X) = \{ \sigma_{u_1}, \sigma_{u_2}, \ldots, \sigma_{u_N} \} \), where \( \sigma_{u_i} = \sigma_{u_i}' \) for all \( u_i \neq u_j \) and \( \sigma_{u_i} \) is such that \( \sigma_{u_i}(k) = \sigma_{u_i}'(k) \) for all \( k \in \{1, 2, \ldots, K - 1\} \) and \( \sigma_{u_i}(K) = (o, t) \). That is, \( \mathcal{P}(X) \) is the same as \( \mathcal{P}(X') \) except that in \( \mathcal{P}(X) \) the atomic objective \( o \) is assigned to UAV \( u_i \) to be executed at time \( t \). The execution time \( t \) is computed as follows. Recall that \( \tau_{\sigma_{u_i}} \) denotes the completion time of the schedule \( \sigma_{u_i} \). The execution time, \( t_i \), is the smallest time that is greater than both \( \tau_{\sigma_{u_i}} \) and the maximum execution time of any predecessor of \( o \) that is
proceeds with the aforementioned four phases, which are detailed in the following four sections.

Given a chromosome $X$ with $|X| = K$, let $X_1, X_2, \ldots, X_{K-1}$ be chromosomes such that for all $k \in \{1, 2, \ldots, K-1\}$, $|X_k| = k$ and $X_k(i) = X(i)$ for all $i \in \{1, 2, \ldots, k\}$ hold. Algorithmically, $\mathcal{PC}(X_1)$ can be computed easily and $\mathcal{PC}(X_k)$ can be computed using $\mathcal{PC}(X_{k-1})$. Finally, $\mathcal{PC}(X)$ can be constructed from $\mathcal{PC}(X_{K-1})$. Notice that this procedure takes linear time with respect to the size of the chromosome.

In the rest of this section detailed discussions of random chromosome generation as well as the other four phases of the GA are provided followed by a formal presentation of the GA.

1. Random Chromosome Generation

Notice that generating chromosomes at random cannot be accomplished by solely picking a random sequence of atomic objectives, since each chromosome must be a trace of the given specification $p_{\text{spec}}$. In this section, we provide an algorithm, which randomly generates a chromosome, i.e., a trace of the specification, such that there is a strictly positive probability for each such trace $\gamma \in \Gamma_p$ to be chosen. The algorithm relies on the $\text{Next} : \mathcal{T} \rightarrow 2^{\mathcal{T} \times \mathcal{O}}$ operator, which maps a given term $p$ to the largest set of pairs $(t', o')$ of terms and atomic objectives, where each $(t', o')$ in this set are such that $p \overset{o'}{\rightarrow} t'$ holds. An algorithmic procedure to compute $\text{Next}(p)$ for a given specification $p$ is provided in Algorithm 1. Notice that this algorithm requires time that is only linear with respect to the size of the specification.

**Algorithm 1: Computation of $\text{Next}(p) : \mathcal{T} \rightarrow 2^{\mathcal{T} \times \mathcal{O}}$ for given $p \in \mathcal{T}$**

1. switch $p$ do
2.   case $p_1 + p_2$
3.     return $\text{Next}(p_1) \cup \text{Next}(p_2)$
4.   case $p_1 \cdot p_2$
5.     return $\{(p'_1 \cdot p_2, t') \mid (p'_1, t') = \text{Next}(p_2)\}$
6.   case $p_1 \parallel p_2$
7.     return $\{(p'_1 \parallel p_2, t') \mid (p'_1, t') = \text{Next}(p_1)\} \cup$
8.     $\{(p_1 \parallel p'_2, t') \mid (p'_2, t') = \text{Next}(p_2)\}$
9.   otherwise
10.    return $\{((\sqrt{}, y))\}$
11. end

Starting with a specification $p$, Algorithm 1 can be used to generate a random next atomic objective $o'$ and a corresponding $t'$ such that $p \overset{o'}{\rightarrow} t'$. This procedure can be continued with $t'$, instead of $p$ to generate a sequence of atomic objectives, until the terminated process $\sqrt{}$ is reached. Then, the resulting sequence of atomic objectives is a valid chromosome. This procedure is presented in a more formal way in Algorithm 2, where $\text{Rand}(S)$ is a primitive function, which returns an element of a given set $S$ at random (according to a uniform distribution among the elements of $S$).

**Algorithm 2: RandomGenerate($p$) : $\mathcal{T} \rightarrow \Gamma_p$**

1. if $p = \sqrt{}$ then
2.   return $\Lambda$
3. else
4.   $S := \text{Next}(p)$
5.   $(t', o') := \text{Rand}(S)$
6.   $\gamma' := \text{Randomgenerate}(p')$
7.   return $o'\gamma'$
8. end

Note that Algorithm 2 return exactly one random trace of $p$. Using Algorithm 2 repeatedly, however, one can generate a set $\mathcal{X}$ of chromosomes to initialize the GA. After the initial set $\mathcal{X}$ of chromosomes is generated, the GA proceeds with the aforementioned four phases, which are detailed in the following four sections.
2. Selection

In the selection phase, two chromosomes are selected randomly from the set \( \mathcal{X} \) of all chromosomes. The randomization is biased so that those chromosomes that are more fit are selected more easily to be parents. The fitness of a chromosome is defined using the cost of its corresponding schedule as follows:

\[
f_{X_i} = \frac{1}{J_i(\mathcal{P}(X_i))},
\]

where \( i = 1, 2 \) (see Equation (1)). That is, the chromosomes with corresponding schedules of lower cost are rated as more fit ones.

After the selection phase, a child chromosome is produced from these two parent chromosomes via the crossover operation.

3. Crossover

The crossover operation generates a child chromosome \( X' \) from two given parent chromosomes \( X_1 \) and \( X_2 \). Informally speaking, given two parent assignments \( X_1 \) and \( X_2 \), the crossover operation first identifies two distinct sets of atomic objectives: \( S_1 \) and \( S_2 \), where \( S_1 \cup S_2 = \emptyset \) and \( S_1 \cap S_2 = \emptyset \). Then, two different sequences are formed: \( \sigma_1 \) and \( \sigma_2 \), where \( \sigma_1 \) is the order preserving projection of \( X_1 \) onto the set \( S_i \) for \( i = 1, 2 \). In the end, the new child assignment is solely the concatenation of \( \sigma_1 \) and \( \sigma_2 \).

Before formalizing the crossover algorithm, let us first identify two useful functions: \( \text{ChildrenAO}(n) \) and \( \text{RightMostChild}(n) \), where \( n \) is a node in the parse tree of \( p_{\text{spec}} \). The function \( \text{ChildrenAO}(n) \) takes a node \( n \) of the parse tree of \( p \) and returns the set of all atomic objectives that are labels of the leaf nodes of the tree rooted at \( n \). An algorithmic procedure for computing \( \text{ChildrenAO}(n) \) is given in a recursive form in Algorithm 3. The function \( \text{RightMostChild}(n) \) returns the rightmost element of the tree rooted by node \( n \). This function is presented in an algorithmic form in Algorithm 4.

As mentioned earlier, the crossover algorithm first creates two distinct sets \( S_1 \) and \( S_2 \) of atomic objectives, such that \( S_1 \cap S_2 = \emptyset \) and \( S_1 \cup S_2 = \mathcal{O} \). The sets \( S_1 \) and \( S_2 \) are, indeed, formed using a natural ordering of the atomic objectives, which comes from the parse tree itself. Intuitively, when the parse tree is drawn, there is an ordering \( \prec_o \) of atomic objectives such that \( o_1 \prec_o o_2 \) if and only if \( o_1 \) is to the left of \( o_2 \) in the tree. This idea can be formalized by using two functions, \( \text{Left}(o) \) and \( \text{Right}(o) \). Given an atomic objective \( o \in \mathcal{O} \), let the functions \( \text{Left}(o) : \mathcal{O} \mapsto 2^\mathcal{O} \) and \( \text{Right}(o) : \mathcal{O} \mapsto 2^\mathcal{O} \) return the set of atomic objectives that are to the left of \( o \) and right of \( o \) respectively. More precisely, let \( \sigma_n^o \) be the ordered sequence of children of node \( n \), i.e., \( \sigma_n^o = \text{Children}(n) \); then, \( \text{Left}(o) \) is formally defined as follows: \( \hat{o} \in \text{Left}(o) \) if and only if there exists \( n, m_1, m_2 \in \mathcal{N}_p \) such that

- \( m_1, m_2 \in \text{Children}(n) \),
- \( \hat{o} \in \text{ChildrenAO}(m_1) \) but \( \hat{o} \notin \text{ChildrenAO}(m_2) \),
- \( o \in \text{ChildrenAO}(m_2) \) but \( o \notin \text{ChildrenAO}(m_1) \),
- \( \sigma_n^{-1}(m_1) < \sigma_n^{-1}(m_2) \).

Similarly, there holds \( \hat{o} \in \text{Right}(o) \) if and only if all these conditions are satisfied, but instead of the last one we have \( \sigma_n^{-1}(m_2) < \sigma_n^{-1}(m_1) \).

To divide the set of atomic propositions into two distinct sets, one can choose an atomic objective \( o \), and let \( S_1 := \text{Left}(o) \cup \{o\} \) and \( S_2 := \text{Right}(o) \), which can be shown to satisfy \( S_1 \cap S_2 = \emptyset \) and \( S_1 \cup S_2 = \mathcal{O} \). Indeed, to perform the crossover one could just pick an atomic objective \( o \) at random to be the cut point and form the child chromosome accordingly using the sets \( S_1 \) and \( S_2 \) as defined above. However, such an approach does not necessarily yield a child chromosome with a feasible assignment. In order to make the child chromosome feasible for all possible crossover operations, we define a procedure for selecting an atomic objective as the cut point. This procedure first selects an atomic objective \( o_{\text{rand}} \) at random, and uses it if the resulting child chromosome yields a feasible assignment. If not, it selects the “nearest” atomic proposition, to the right of \( o_{\text{rand}} \), that would yield a feasible assignment. This procedure is given in Algorithm 5.
must also be a valid chromosome, i.e., a trace of the specification \( p \) modified to an other chromosome \( X \). In the beginning of the mutation phase, a chromosome

The mutation operation is used for making random changes in a small part of the generation so as to avoid local minima. In the beginning of the mutation phase, a chromosome \( X \) is selected from the set \( \mathcal{X} \) of chromosomes and modified to an other chromosome \( X' \). This operation, however, is not very trivial, since the modified chromosome \( X' \) must also be a valid chromosome, i.e., a trace of the specification \( p_{\text{spec}} \).

The mutation phase proceeds as follows. Firstly, a number of chromosomes are picked at random from the set \( \mathcal{X} \) of chromosomes. Secondly, for each such chromosome \( X \), an atomic objective \( o_{\text{mid}} \) is picked at random which

<table>
<thead>
<tr>
<th>Algorithm 3: ChildrenAO(n): Atomic Objectives connected to ( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 if leaf(( n )) = true then</td>
</tr>
<tr>
<td>2 ( S = { n } )</td>
</tr>
<tr>
<td>3 else</td>
</tr>
<tr>
<td>4 ( \sigma := \text{children}(n) )</td>
</tr>
<tr>
<td>5 ( S := \emptyset )</td>
</tr>
<tr>
<td>6 for ( i = 1 ) to (</td>
</tr>
<tr>
<td>7 ( S := S \cup \text{childrenAO}(\sigma(i)) )</td>
</tr>
<tr>
<td>8 end</td>
</tr>
<tr>
<td>9 end</td>
</tr>
<tr>
<td>10 return ( S )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm 4: RightMostChild(n): Right most child of the tree rooted by ( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 while leaf(( n )) = false do</td>
</tr>
<tr>
<td>2 ( \sigma := \text{children}(n) )</td>
</tr>
<tr>
<td>3 ( n := \sigma(</td>
</tr>
<tr>
<td>4 end</td>
</tr>
<tr>
<td>5 return ( n )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm 5: cutAtomicObjective(( X_1, X_2 )) : ( \Gamma_{p_{\text{spec}}} \times \Gamma_{p_{\text{spec}}} \rightarrow \mathcal{O} ): Generate a cutting point AO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( o_{\text{rand}} = \text{rand} )</td>
</tr>
<tr>
<td>2 ( S := { o_{\text{rand}} } )</td>
</tr>
<tr>
<td>3 while ( (o \notin X_1 \text{ for } \forall o \in S) ) or ( (o \notin X_2 \text{ for } \forall o \in S) ) do</td>
</tr>
<tr>
<td>4 ( \text{while Operator}(n) = + ) do</td>
</tr>
<tr>
<td>5 ( n := \text{parent}(n) )</td>
</tr>
<tr>
<td>6 end</td>
</tr>
<tr>
<td>7 ( S := \text{childrenAO}(n) )</td>
</tr>
<tr>
<td>8 end</td>
</tr>
<tr>
<td>9 if leaf(( n )) = false then</td>
</tr>
<tr>
<td>10 ( n := \text{rightMostChild}(n) )</td>
</tr>
<tr>
<td>11 end</td>
</tr>
<tr>
<td>12 ( o_{\text{cut}} := \text{AtomicObjective}(n) )</td>
</tr>
<tr>
<td>13 return ( o_{\text{cut}} )</td>
</tr>
</tbody>
</table>

Given a cut point atomic objective \( o_{\text{cut}} \), the sets \( S_1 := \text{left}(o_{\text{cut}}) \cup \{ o_{\text{cut}} \} \) and \( S_2 := \text{right}(o_{\text{cut}}) \) can be computed. Then, the aforementioned sequences \( \sigma_1 \) and \( \sigma_2 \) are the order preserving projections of \( X_1 \) and \( X_2 \) to the sets \( S_1 \) and \( S_2 \) respectively, i.e., \( \sigma_i = [X_i]_{S_i} \) for \( \forall i \in \{1, 2\} \).

The crossover operation is summarized in Algorithm 6. Given two chromosomes \( X_1 \) and \( X_2 \), the crossover algorithm first generates a cut point atomic objective \( o_{\text{cut}} \) via Algorithm 5. In the second step, it generates the two sets \( S_1 := \text{left}(o_{\text{cut}}) \cup \{ o_{\text{cut}} \} \) and \( S_2 := \text{right}(o_{\text{cut}}) \), which represent the set of atomic objectives to the left of \( o_{\text{cut}} \) and the ones to the right of \( o_{\text{cut}} \), respectively. Using these two sets, two sequences, \( \sigma_1 \) and \( \sigma_2 \), are generated from chromosomes \( X_1 \) and \( X_2 \). Finally, the resulting child chromosome \( X' \) is the concatenation of the two sequences \( \sigma_1 \) and \( \sigma_2 \).

<table>
<thead>
<tr>
<th>Algorithm 6: Crossover(( X_1, X_2 )) : ( \Gamma_{p_{\text{spec}}} \times \Gamma_{p_{\text{spec}}} \rightarrow \Gamma_{p_{\text{spec}}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( o_{\text{cut}} := \text{cutAtomicObjective}(X_1, X_2) )</td>
</tr>
<tr>
<td>2 ( S_1 := \text{left}(o_{\text{cut}}) \cup { o_{\text{cut}} } )</td>
</tr>
<tr>
<td>3 ( S_2 := \text{right}(o_{\text{cut}}) )</td>
</tr>
<tr>
<td>4 ( \sigma_1 := [X_1]_{S_1} )</td>
</tr>
<tr>
<td>5 ( \sigma_2 := [X_2]_{S_2} )</td>
</tr>
<tr>
<td>6 return ( \sigma_1 \sigma_2 )</td>
</tr>
</tbody>
</table>

It is not clear whether the procedure yields a child chromosome that is indeed a trace of the given specification. We postpone this discussion to address later in the paper and continue with the other genetic algorithm operations.

4. Mutation

The mutation operation is used for making random changes in a small part of the generation so as to avoid local minima. In the beginning of the mutation phase, a chromosome \( X \) is selected from the set \( \mathcal{X} \) of chromosomes and modified to an other chromosome \( X' \). This operation, however, is not very trivial, since the modified chromosome \( X' \) must also be a valid chromosome, i.e., a trace of the specification \( p_{\text{spec}} \).
satisfies $o_{mid} \in X$. Let $\sigma', \sigma'' \in \Sigma_O$ be such that $\sigma'|\sigma'' = X$ and $\sigma''(1) = o_{mid}$. Moreover, let $p' \in \mathbb{T}$ be such that $p_{spec} \xrightarrow{\sigma'} p'$. Finally, the mutated chromosome $X'$ is computed using Algorithm 2 as follows. Let $\sigma_{rand} = \text{RandomGenerate}(p')$; then, $X' = \sigma'|\sigma_{rand}$. The mutation phase is formalized in Algorithm 7.

### Algorithm 7: $\text{mutate}(X) : \Gamma_{p_{spec}} \rightarrow \Gamma_{p_{spec}}$; Mutate a given chromosome $X$

1. $o_{mid} := \text{Rand}(\{o \in \mathcal{O} | o \in X\})$
2. Let $\sigma', \sigma'' \in \Sigma_O$ be such that $p_{spec} = \sigma'|\sigma''$ and $\sigma''(1) = o_{mid}$
3. Let $p'$ be such that $p_{spec} \xrightarrow{\sigma'} p'$
4. $\sigma_{rand} := \text{RandomGenerate}(p')$
5. return $\sigma'|\sigma_{rand}$

5. **Elitism**

In the elitism phase, the chromosomes with high fitness are selected to move into the next generation. The selection is again made randomly, however, biased towards the higher fitness values of the chromosomes. Though, we deterministically choose a small set of chromosomes of the current generation with the highest fitness values, elite chromosomes, to maintain, as it rules out the possibility of loosing all the good solutions in a given generation. This provides us with a solution that is monotonically improving.

### B. Algorithm

Let $\text{Rand}(S, value, k)$ be a primitive function, where $S$ is a finite set, $value : S \rightarrow \mathbb{R}$ is a function, and $k$ is number such that $k < |S|$. The function $\text{Rand}(S, value, k)$ returns a set of $k$ distinct elements from $S$ such that an element $s \in S$ is returned with probability $value(s)/\sum_{s' \in S} value(s')$. Let also $\text{SelectBest}(S, value, k)$ be another primitive function that returns a set of $k$ elements from set $S$ such that for all $s \in \text{SelectBest}(S, value, k)$ and for all $s' \in S \setminus \text{SelectBest}(S, value, k)$ we have that $value(s) \geq value(s')$.

The GA is formalized in Algorithm 8, where the initialization phase (Lines 8-8) as well as the selection (Lines 8-8), crossover (Lines 8-8), mutation (Lines 8-8), and elitism (Line 8) operations are shown explicitly.

### Algorithm 8: $(p_{spec}, K_{total}, K_{children}, K_{elite}, K_{mutate}, N)$

1. $\mathcal{X} := \emptyset$
2. for $i := 1$ to $K_{total}$ do
   3. $\mathcal{X} := \mathcal{X} \cup \text{RandomGenerate}(p_{spec})$
3. end
4. for $j := 1$ to $N$ do
   5. for all $X_k \in \mathcal{X}$ do
      6. $\text{fitness}(X_k) := 1/J(\text{PC}(X_k))$
   7. end
8. $\mathcal{C} := \emptyset$
9. for $i := 1$ to $K_{children}$ do
   10. $X_1 := \text{Rand}(\mathcal{X}, \text{fitness}, 1)$
    11. $X_2 := \text{Rand}(\mathcal{X}, \text{fitness}, 1)$
    12. $C := \text{Crossover}(X_1, X_2)$
    13. $\text{fitness}(C) := 1/J(\text{PC}(C))$
    14. $\mathcal{C} := \mathcal{C} \cup C$
15. end
16. $\mathcal{M} := \emptyset$
17. for $i := 1$ to $K_{mutate}$ do
   18. $X := \text{Rand}(\mathcal{X}, \text{fitness}, 1)$
   19. $M := \text{Mutate}(X)$
   20. $\text{fitness}(M) := 1/J(\text{PC}(C))$
   21. $\mathcal{M} := \mathcal{M} \cup M$
22. end
23. $\mathcal{X} := \text{SelectBest}(\mathcal{X}, K_{elite}) \cup \text{Rand}(\mathcal{X}, C, M, \text{fitness}, K_{total} - K_{elite})$
24. end
25. return $\text{SelectBest}(\mathcal{X}, 1)$
Next we discuss some interesting properties of the algorithm. Firstly, notice that the \( \text{RandomGenerate}(p_{\text{spec}}) \) function presented in Algorithm 2 is correct in the sense that it returns only those traces that are in \( \Gamma_{p_{\text{spec}}} \). This fact follows from the correctness of the \( \text{Next} \) function, for which an algorithmic procedure was presented in Algorithm 1. Let us show that each trace in \( \Gamma_{p_{\text{spec}}} \) is selected by \( \text{RandomGenerate}(p_{\text{spec}}) \) with probability strictly greater than zero.

**Proposition V.1** Given any specification \( p_{\text{spec}} \in \mathbb{T} \) defined on a given set of atomic objectives \( \mathcal{O} \), the probability that 
\[
\gamma = \text{RandomGenerate}(p_{\text{spec}})
\]

is at least \( 1/|\mathcal{O}|^{|\mathcal{O}|} \).

Next, let us show that the crossover operation is correct in the sense that \( \text{Crossover}(X_1, X_2) \) returns a valid chromosome given that \( X_1 \) and \( X_2 \) are valid chromosomes.

**Proposition V.2** Given any specification \( p_{\text{spec}} \in \mathbb{T} \) and any two chromosomes \( X_1 \) and \( X_2 \) such that \( X_1, X_2 \in \Gamma_{p_{\text{spec}}} \), then we have that \( \text{Crossover}(X_1, X_2) \in \Gamma_{p_{\text{spec}}} \).

Let us also note that through mutation operation one can obtain any valid chromosome with some strictly positive probability. More formally, we state the following proposition.

**Proposition V.3** Given any specification \( p_{\text{spec}} \in \mathbb{T} \) defined on a given set of atomic objectives \( \mathcal{O} \) and any chromosome \( X \in \Gamma_{p_{\text{spec}}} \), for any \( X' \in \Gamma_{p_{\text{spec}}} \) the probability that \( X' = \text{Mutate}(X) \) is at least \( 1/|\mathcal{O}|^{|\mathcal{O}|} \).

### VI. Simulations

A simulation study is presented in this section. All the simulations are performed on a laptop computer with 2.5 GHz processor clock speed and 4GB RAM running Ubuntu Linux operating system; the algorithm is implemented using C++ and its standard libraries.

Motivated by a perimeter security application, consider a military UAV surveillance mission, in which five different targets distributed over a square region have to be (i) classified and (ii) destroyed in this order by using one of the two UAVs for each task. Each target can be engaged independently, i.e., there is no temporal or logical dependence between the targets. A map of the scenario is shown in Figure 1, where the positions of the targets can be observed (red circles) and all the UAVs start flying from the same location (blue circle). Both the classification and destruction task require 0.1 hours to perform each. Let \( \mathcal{U} = \{u_1, u_2\} \) denote the set of UAVs. In this scenario, the UAVs \( u_1 \) and \( u_2 \) can navigate with 20 mph and 30 mph top speeds, respectively. The objective is to find a schedule for each one of the UAVs such that the mission time is minimized (i.e., the cost function is \( J_2 \) given in Equation (1)).

![Figure 1. Map of the military UAV mission.](image)

This problem can be modeled in the process algebra framework as follows. Let us define the atomic objectives \( o_{i,j,u} = (x_{o_{i,j,u}}, \pi_{o_{i,j,u}}, T_{o_{i,j,u}}, U_{o_{i,j,u}}) \), where the indices \( i, j, u \) range over the following sets: \( i \in \{1, \ldots, 5\} \), \( j \in \{c, d\} \), \( u \in \{u_1, u_2\} \). Intuitively, \( o_{i,j,u} \) refers to the atomic objective of performing task \( j \in \{c, d\} \), denoting classification and destruction respectively, on target \( i \) via UAV \( u \). Hence, the set \( \mathcal{O} \) of atomic objectives includes a total of twenty elements.

The temporal and logical constraints of the problem can be formulated in the process algebra specification framework as follows. Let us denote the objective of classifying the \( i \)th target as \( p_{i,c} \). Notice that, since this classification operation can be handled by either UAV \( u_1 \) or UAV \( u_2 \), this objective can be represented by the process algebra term \( p_{i,c} = o_{i,c,u_1} + o_{i,c,u_2} \). Similarly, for the destruction objective on target \( i \), we have \( p_{i,d} = o_{i,d,u_1} + o_{i,d,u_2} \). Recall that each target must be first classified and destroyed in this order. Denoting the objective of engaging target \( i \) with
Finally, noting that all the targets can be engaged independently, we have the following:

\[ p_{\text{spec}} = p_1 \parallel p_2 \parallel p_3 \parallel p_4 \parallel p_5, \]

where \( p_{\text{spec}} \) denotes the main objective.

Simulation results for three different sets of parameters are shown in Figure 2. The parameter sets 1, 2, and 3 assign the parameters \( \{K_{\text{total}}, K_{\text{children}}, K_{\text{elite}}, K_{\text{mutation}}\} \) the values \( \{10, 5, 3, 2\} \), \( \{100, 85, 10, 5\} \), and \( \{1000, 900, 50, 50\} \). As seen from the figure, the algorithm takes 3 iterations to converge to the optimum value with parameter set 3; with parameter set 2, on the other hand, it takes the algorithm 18 iterations to converge to the optimum solution. With parameter set 1 the convergence is quite slow. Note that optimality of the solution was verified using a tree search algorithm. In Figure 3, the cumulative computation time for each iteration is shown. Note that the time of the first iteration includes that of the initialization step as well.

In order to demonstrate the computational properties of the algorithm large-scale scenarios are also considered. The cost for same three sets of parameters as well the corresponding cumulative computation time for similar scenarios, but this time with 3 UAVs and 10, 20, 30, 40, and 50 targets (where two tasks have to be executed in each target) are shown in Figure 4. The number of atomic objectives ranges from 60 to 300 in these scenarios. Target locations are randomly selected for each set of simulations.

![Figure 2](image2.png)

**Figure 2.** The mission time corresponding to the best chromosome in the population versus the iteration no. Parameter sets 1, 2, and 3 are shown in red, blue, and green, respectively. The optimal value is 1.407, which is achieved by parameter sets 2 and 3.

![Figure 3](image3.png)

**Figure 3.** Cumulative computation time for each parameter set.

### VII. Conclusions

A process algebra based genetic algorithm was presented for solving complex mission planning problems for multiple UAVs. The approach was demonstrated on an example UAV base defence problem. It was shown that process algebra can be used to formally define the evolutionary operators of crossover and mutation. The viability of the approach was investigated using simulations of small and large scale problems. It was shown that, for a small sized problem, the algorithm converges to the optimal solution.
Acknowledgments

This research was partially supported by the Michigan/AFRL Collaborative Center on Control Sciences, AFOSR grant no. FA 8650-07-2-3744. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the supporting organizations.

References


Figure 4. Simulations results for up to 300 atomic objectives. Parameter sets 1, 2, and 3 are shown in red, blue, and green, respectively. Number of targets are shown on the subfigure captions.