Constrained Task Assignment and Scheduling On Networks of Arbitrary Topology

by

Justin Patrick Jackson

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Aerospace Engineering)
in The University of Michigan
2012

Doctoral Committee:

Assistant Professor Anouck Renee Girard, Chair
Professor Edmund H. Durfee
Professor Pierre Tshimanga Kabamba
Mariam Faied Abdelhafiz

To all of my parents and mentors, thank you for everything.

ACKNOWLEDGEMENTS

Thank you...

To my advisor Professor Anouck Girard, thank you for always making sure I didn't go hungry. Thank you for excusing and supporting my many interests. It has been my pleasure to work with you and grow as your student. Professor Pierre Kabamba, thank you for your clarity in aspects from moral character to technical things and stuff. Thank you for imparting on me the importance of seeking the same for myself. Mariam Faied, thank you for having the patience to support so many long technical discussions, for reading all of my writing, and trying my cooking. Professor Edmund Durfee, thank you for taking the time to offer your perspective that has helped me to dig deeper and improve my own knowledge. Your work in distributed systems has helped spark my own related interest. Professor Alec Gallimore, thank you for giving me the idea to apply to the University of Michigan, for helping me my first year here, for participating with the Society of Minority Engineers and Scientists - Graduates, and for being a leader in the black community. My colleagues at the Air Force Research Labs, thank you for your support, input, and direction over the years.

Johnhenri Richardson, thank you for being my officemate which often entailed being distracted from your work by my research concerns. My colleagues in the ARC Lab, especially Zahid Hasan, Calvin Park, Yu-hsien Chang, Baro Hyun, Chris Orlowski, and Jon White, thank you for just being great friends from the start. The Society of Minority Engineers and Scientists - Graduates, it was a pleasure to serve

with you, the executive board and the members, during my time at the University of Michigan. Tom Porter and the fellows of the Frankel Fund, thank you first for accepting me as a fellow, second for being a joy to work with and having so much to teach me. The Epeians, thank you for recognizing my leadership tendencies and inducting me into your ranks. It is a source of pride.

Professor Johnny Hurtado at Texas A&M for sparking my interest in dynamics and controls. I am fortunate to have you as a mentor in character as well as technically. Drew Woodbury, one of my truly great friends during my time in graduate school. Lastly and mostly, thank you to all of my parents and family for supporting me through extreme frustration and joy. And simply for being proud of me.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

Constrained Task Assignment and Scheduling On Networks of Arbitrary Topology

by

Justin Jackson

Chair: Anouck Girard

This dissertation develops a framework to address centralized and distributed constrained task assignment and task scheduling problems. This framework is used to prove properties of these problems that can be exploited, develop effective solution algorithms, and to prove important properties such as correctness, completeness and optimality.

The centralized task assignment and task scheduling problem treated here is expressed as a vehicle routing problem with the goal of optimizing mission time subject to mission constraints on task precedence and agent capability. The algorithm developed to solve this problem is able to coordinate vehicle (agent) timing for task completion. This class of problems is NP-hard and analytical guarantees on solution quality are often unavailable. This dissertation develops a technique for determining solution quality that can be used on a large class of problems and does not rely on traditional analytical guarantees.

For distributed problems several agents must communicate to collectively solve a distributed task assignment and task scheduling problem. The distributed task

assignment and task scheduling algorithms developed here allow for the optimization of constrained military missions in situations where the communication network may be incomplete and only locally known. Two problems are developed. The distributed task assignment problem incorporates communication constraints that must be satisfied; this is the Communication-Constrained Distributed Assignment Problem. A novel distributed assignment algorithm, the Stochastic Bidding Algorithm, solves this problem. The algorithm is correct, probabilistically complete, and has linear average-case time complexity.

The distributed task scheduling problem addressed here is to minimize mission time subject to arbitrary predicate mission constraints; this is the Minimum-time Arbitrarily-constrained Distributed Scheduling Problem. The Optimal Distributed Non-sequential Backtracking Algorithm solves this problem. The algorithm is correct, complete, outputs time optimal schedules, and has low average-case time complexity.

Separation of the task assignment and task scheduling problems is exploited here to ameliorate the effects of an incomplete communication network. The mission-modeling conditions that allow this and the benefits gained are discussed in detail. It is shown that the distributed task assignment and task scheduling algorithms developed here can operate concurrently and maintain their correctness, completeness, and optimality properties.

CHAPTER I

Introduction

This dissertation addresses problems of centralized and distributed task assignment and task scheduling. Task assignment is deciding which of several agents will perform which of several tasks. Task scheduling is the act of deciding at what time a task is performed. Task assignment and task scheduling can be done in three ways: first, the needed data can be collected by the agents and aggregated by a central computer which then designs the task assignment and task schedule and distributes orders to the agents. Second, the needed data can be aggregated by each of the agents; the same task assignment and task schedule is designed by each agent. Third, the needed data can be collected by the agents, but is used by the agents to each compose a part of the task assignment and task schedule and is never aggregated. This dissertation addresses the first and third of these. The first is referred to as centralized task assignment and task scheduling, and the third as distributed task assignment and task scheduling. The remainder of this chapter discusses these concepts in further detail, presents a preliminary problem statement, and discusses the contributions of this dissertation to the fields of centralized and distributed task assignment and task scheduling.

1.1 Motivation and Importance of Task Assignment and Task Scheduling

Task assignment and task scheduling are essential in applications such as truck and car dispatching, load balancing for the efficient management of computational resources, and recently in the management and optimization of military missions involving unmanned aircraft. Task assignment and task scheduling are often considered together and the solution is computed in a centralized way, that is, by a central computer, and then communicated to the agents. In applications where all agents can communicate with each other and with the central computer, this approach may be acceptable. In applications where this is not the case or where the agents have significant computational abilities that can be leveraged, improvements in robustness, system response to changes in data, and computation time can be gained by distributing the problem. This dissertation considers both centralized and distributed task assignment and task scheduling.

One of the aims of this dissertation is to provide tools that help advance the operational vision of the U.S. Air Force as detailed in [24]. This vision includes the ability to integrate unmanned vehicles into Air Force operations. These vehicles should be able to utilize their computational capabilities to function independently of and concurrently with their human counterparts. This vision represents an ongoing research effort at the Air Force Research Laboratories and complementary organizations. Several features are commonly incorporated in problems within the research effort. Mobile agents are considered that are kinematically constrained and typically travel within a two dimensional environment. Mission constraints are used to restrict how a mission can be executed by the agents. These constraints must be obeyed when deciding the task assignment and the task schedule. In these applications, task assignment and task schedules are sought that minimize an objective function related

to mission performance. The Air Force's vision is one of autonomy; the agents should be able to utilize their computational and communication abilities to design task assignments and task schedules that obey these constraints and provide good mission performance. The reality of field operations includes constraints on the capabilities of the agents and limited or unreliable communications. The need to assign and schedule mission-related tasks to agents in an optimized way under these conditions is a primary motivation of this dissertation.

The need for unmanned air and ground vehicles to function independently and effectively with their human counterparts necessitates local intelligence on-board the vehicles. The vehicles must be able to gather and process information and perform tasks without the intervention of human operators. For certain missions, the human may need to remain more abreast of the situation and have a finer level of control on the individual tasks (e.g., remote piloting). For other missions, the vehicles may be able to assume more control over the individual tasks (e.g., cooperative area mapping).

This thesis addresses the part of this problem that deals with assigning and scheduling tasks after the tasks themselves have been decided upon. This work stops before the execution of the tasks takes place; it does not consider execution inter-woven with the task assignment and task scheduling process. The reduction of man-power requirements for supporting autonomous vehicles, improved mission optimization, and the certifiability of autonomous systems operating with provably correct algorithms are among the reasons for the widespread development of mission optimization algorithms. Mission aspects such as vehicle failure, communication jamming, communication blackouts, and the use of heterogeneous communication protocols challenge the ability of the vehicles to assign and schedule tasks independently. Distributed task assignment and task scheduling offers the ability to achieve these mission optimization goals when effectiveness of communications and the agents themselves is reduced.

An important objective considered in both the centralized and distributed work here is the minimization of mission time (also know as makespan). This objective represents the time needed for the agents to complete all of the tasks given. This objective function is important when considering urgency, applied to the mission as a whole. It is not an *a priori* weighted sum of the start or finish times of the tasks. It may be desired that individual tasks be completed quickly; a portion of this dissertation addresses this, but the minimization of mission time is the primary objective.

Task assignment and task scheduling fit into the broader class of work on multi-agent systems [118]. Task assignment and task scheduling are important when multiple agents must complete a common goal that is composed of several sub-goals; these sub-goals are tasks. In practical situations where task assignment and task scheduling are required such as military missions or disaster relief, the relative (or absolute) timing between tasks, the decision to complete tasks, and which tasks agents are permitted to perform, may be relatively constrained. The effect of these types of constraints is to limit the allowable task assignments and task schedules. Task assignments and task schedules that obey the relevant constraints are said to be feasible. In addition to obeying these practical restrictions it is often of interest to minimize the use of valuable resources such as fuel or time. Feasible task assignments and task schedules that also achieve this minimization are said to be optimal.

1.2 Problem Statement

In this section, we introduce the formal concepts used for task assignment and task scheduling throughout. We capture the various notions of feasibility, optimality, and communication using set theory and graph theory; we use tools from combinatorial optimization to describe tasks, agents, the ways in which they interact with each other, and to develop solution tools. This rigor aids in the presentation of the problem

and development of the proofs.

There are N_t tasks and N_a agents. The tasks and agents are elements of the sets

$$\mathcal{T} = \{t_1, \ldots, t_{N_t}\}, \tag{1.1}$$

$$\mathcal{A} = \{a_1, \ldots, a_{N_a}\}. \tag{1.2}$$

A task assignment can take the form of a mapping,

$$TA : \mathcal{T} \to \mathcal{A}, \tag{1.3}$$

or a relation

$$TA \subseteq \mathcal{T} \times \mathcal{A}. \tag{1.4}$$

A mapping task assignment implies that each task is performed by one and only one agent while a relational task assignment implies that several agents can cooperate to perform a single task.

A task schedule takes the form of a mapping,

$$TS : \mathcal{T} \to \mathbb{T}_s, \tag{1.5}$$

where the set $\mathbb{T}_s$ is the set of schedule times. Generally, these times can represent the time when a task starts or finishes execution.

The timing of performing tasks is restricted logically and temporally. This is described using constraints of the form $p : \mathbb{T}_s^{\mathcal{T}} \to \{false, true\}$. These constraints are general to allow for the use of numerous constraint description languages. The tasks involved in these constraints belong to clusters. These clusters are the sets $\mathcal{T}_m \subseteq \mathcal{T}$, $m = 1, \ldots, N_c$.

The ability of the agents to complete the tasks is described using a relation $Capability \subseteq \mathcal{T} \times \mathcal{A}$ where $(t, a) \in Capability$ if task t can be performed by agent a.

It is important that task assignments obey $(t, TA(t)) \in Capability$ for all tasks and that task schedules obey $p_m(TS) = true$ for all constraints.

Much of this dissertation is concerned with optimization of mission time. The objective function is

$$J(TS) = \max_{t \in \mathcal{T}} TS(t). \tag{1.6}$$

The task assignment literature offers tools to optimize task assignments with respect to linear and quasi-linear objective functions [80, 7, 116, 100]. These objective functions often represent distance, monetary value, or time. We minimize the time to complete all tasks by solving

$$\min_{TS \in \mathbb{T}_s^{\mathcal{T}}} J(TS) \tag{1.7}$$

$$\text{s.t. } p_m(TS). \tag{1.8}$$

The approach taken here to solving the optimal task assignment and task scheduling problem is to develop a heuristic approach that optimizes task assignments and task schedules simultaneously. This approach is novel in that it incorporates multiple agents and precedence constraints with the minimization of mission time, and can find feasible solutions quickly and low cost solutions in minutes. This type of optimization is important to military and other time-critical applications.

1.3 Distributed Task Assignment and Task Scheduling

A distributed system is a collection of agents in which resources, information, knowledge, capability, expertise, or authority are distributed. In the context of distributed task assignment and task scheduling we consider as distributed: computational resources; information and knowledge of the tasks, agent capabilities, constraints, and the communication network topology; and authority of agents to determine the task assignment and task schedule.

It can be beneficial to exploit parallelism and distribute the computational burden. Additionally, this is beneficial because the solution must be distributed (in some form) prior to execution. Communication between certain pairs of agents (including a possible central planner) may be sporadic, non-existent, unsecured, or delayed. For these reasons we consider distributed task assignment and task scheduling.

For multi-agent systems, which agents can communicate with each other is important. Communication can be used to distribute the effort of determining the task assignment and task schedule across several agents. Candidate algorithms for task assignment and scheduling are impacted differently as a result of different assumptions on the topology of the underlying communication network. This dissertation considers some of these issues and provides algorithms to address the problem of a connected arbitrary network topology.

To describe the communication abilities of the agents, we use an undirected communication graph $(\mathcal{A}, \mathcal{E}_c)$, where the edges, $\mathcal{E}_c$, represent acknowledgement-based communication links. Messages are received in the order they are sent and when the communication link is established, the messages are guaranteed to be delivered. If messages are broadcast or relayed across the network, they are done so without acknowledgement between the source and the recipient. The effect of this is that messages can be relayed reliably, but are relayed without guaranteeing the delivery order. This model of communication is common in distributed consensus applications [93]. This model allows us to incorporate the fact that in field or ad-hoc situations, acknowledgement-based communication may not be guaranteed between all pairs of agents. This can occur in situations where mobile robots may operate in applications of disaster relief, aircraft and ground vehicles may operate in canyons or mountainous terrain, or communication may be subject to black-out or jamming.

It is important for us to do the task assignment and task scheduling in the presence of various types of constraints. Existing work has addressed distributed assignment to

optimize various objectives. Formulations and algorithms exist for addressing capability constrained task assignment problems, and temporally constrained task scheduling problems [33, 58, 34]. Our concern is for temporally and logically constrained distributed task assignment and task scheduling. The agents in the following formulations cooperate and seek to optimize a global objective. The notion of local benefit is used only as a means to achieve the global goal. To ensure that all agents assigned tasks that share a constraint can communicate, we solve the problem of finding a task assignment, TA, that, for all $\mathcal{T}_m \subseteq \mathcal{T}$, satisfies

$$(\mathcal{A}, \mathcal{E}_c)|_{TA(\mathcal{T}_m)} \text{ is complete.} \tag{1.9}$$

That is, agents can communicate with all agents assigned tasks that share clusters with their own tasks. The Stochastic Bidding Algorithm used to satisfy the clustering constraints is designed to minimize an objective function, $J : \mathcal{A}^{\mathcal{T}} \rightarrow \mathbb{N}$, with the following properties. The value of J goes to zero as more of the constraints in (1.9) become satisfied, and $J(TA) = 0$ if and only if all constraints in (1.9) are satisfied. This objective function is nonlinear, has a global minimum at zero, and is discussed in further detail in Section 4.1.2.

The satisfaction of (1.9) is used to solve the above scheduling problem in a distributed setting. The Optimal Distributed Non-Sequential Backtracking Algorithm is developed here to ensure minimization of the mission time objective function while guaranteeing constraint satisfaction. This algorithm is proven correct, complete, and optimal, and complexity results are given in Section 4.2.5.

1.4 Task Assignment and Task Schedule Coupling and Consequences

We are interested in finding task assignments and task schedules that obey constraints and are optimal with respect to mission time. As such, we must consider task assignments and task schedules together. The constraints that describe the sets of allowable task assignments and task schedules can couple the two. This dissertation describes, for the distributed case, a relationship between the expressiveness of the problem constraints and the communication links required to solve the problem (Section 4.1.7). This relationship is exploited here to solve the distributed task assignment and task scheduling problem.

The following consequences are incurred if existing distributed constraint satisfaction and optimization tools are directly applied to this problem: the number of acknowledgement-base communication links required is increased; the complexity of solving the problem can increase; and in some cases, minimization of mission time subject to the constraints cannot be guaranteed. This is proven in Chapter II. This dissertation presents a formulation to express the above constrained, distributed, task assignment and task scheduling problem and develops distributed tools to solve it.

The algorithms developed here exploit the following aspects of the problem structure. The communication structure is exploited to find areas of the network where the communication topology is suitable for distributed scheduling algorithms. The constraint structure is exploited to separate task assignment and task scheduling constraints and solve the assignment and scheduling problems concurrently. The structure of the scheduling constraints is exploited to solve independent scheduling subproblems concurrently. The structure of the cost function $J(TS) = \max_{t \in \mathcal{T}} TS(t) \in \mathbb{T}_s$ is used to guarantee that the constraints are satisfied while J is minimized.

1.5 Modeling the Distributed System

In modeling the distributed capabilities of the agents, we use the notion of processes [75]. Several areas of the multi-agent literature use the notion of agent to denote a computational entity that executes a distributed algorithm [118]. The formulation of [75] is used here to explicitly differentiate between agents and their capabilities to perform tasks. Using processes allows us to easily distinguish between the agent that is involved in the assignment and the "agent" (i.e., the process) that is logical and performs the computation of the distributed algorithm.

We introduce several standard notions from graph theory to allow us to reason about the communication abilities and restrictions of the agents. An *undirected graph* is a pair $(\mathcal{V}, \mathcal{E})$ of vertices and edges such that each edge is a couple of vertices. The edges have unit distance. A graph is called *complete* if every couple of vertices is an edge. For the graph $(\mathcal{V}, \mathcal{E})$, if $\mathcal{V}' \subseteq \mathcal{V}$, the *subgraph induced by restriction to* $\mathcal{V}'$, denoted $(\mathcal{V}, \mathcal{E}) \mid_{\mathcal{V}'}$, is the graph $(\mathcal{V}', \mathcal{E}')$, where

$$\mathcal{E}' = \{\{v_1, v_2\} \in \mathcal{E} \mid v_1 \in \mathcal{V}' \text{ and } v_2 \in \mathcal{V}'\}. \tag{1.10}$$

In other words, the induced subgraph is obtained by retaining only vertices in $\mathcal{V}'$ and the edges connecting them. The *distance* between two vertices $v, w \in \mathcal{V}$ is $d(v, w)$ and represents the number of edges that must be traversed to move from v to w across the graph. The *diameter* of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is,

$$diam(\mathcal{G}) = \max_{v,w \in \mathcal{V}} d(v, w). \tag{1.11}$$

The *neighborhood* of a vertex $v \in \mathcal{V}$ is the set $\mathcal{N}_v = \{w \in \mathcal{V} \mid e = \{v, w\} \in \mathcal{E}\}$.

Consider a distributed system of $N > 0$ processes. Define a process as a 4-tuple,

$$[p_i] = \{States_i, Start_i, trans_i, msgs_i\}, \tag{1.12}$$

where the set $States_i$ is a possibly infinite set of states of process $[p_i]$ and describes the configuration and memory of $[p_i]$, and the set $Start_i \subseteq States_i$ is the subset of states at which process $[p_i]$ may begin operation. Processes send messages $M \in \mathcal{M}$ to each other. The set of messages $\mathcal{M}$ is closed under union. Hence, without loss of generality ($\mathcal{M}$ is closed under union), processes send one message to each neighboring process at a time. The functions $trans_i$ and $msgs_i$ are defined as

$$trans_i \quad : \quad States_i \times \mathcal{M} \rightarrow States_i, \tag{1.13}$$

$$msgs_i \quad : \quad \mathcal{M} \times States_i \rightarrow \mathcal{M}. \tag{1.14}$$

The set of processes is $Processes$. Define the process graph,

$$\mathcal{G}_p = (Processes, \mathcal{E}_p), \tag{1.15}$$

The set $\mathcal{E}_p$ is the set of couples of processes such that $\{[p_i], [p_j]\} \in \mathcal{E}_p$ if and only if there is a communication link between $[p_i]$ and $[p_j]$. Three fundamental classes of distributed systems are *synchronous*, *asynchronous*, and *partially synchronous*.

Define a unit of time called a *round*, by the execution of the functions $trans_i$ and $msgs_i$ for all $[p_i] \in Processes$. For a synchronous distributed system [75], at each round all processes $[p_i] \in Processes$ execute the function $trans_i$ and then all processes $[p_i] \in Processes$ execute the function $msgs_i$. This lock-step type of operation is an idealization of actual distributed systems that can be imposed in practice. However, this sacrifice does not allow for the system to take advantage of differences in the relative speed of operation of processes. Induction-style proofs are made easier by this assumption.

An asynchronous model of a distributed system [37] makes no assumptions at all concerning the relative timing of process operation, or the delivery order (or timing)

of messages. That is, a process $[p_i] \in Processes$ may compute results and send messages without consideration for the operation of other processes. An asynchronous distributed system can be restricted to behave as a synchronous distributed system. This is done by imposing the requirement that each $[p_i] \in Processes$ execute $trans_i$ at round r only after all round $r-1$ messages are received from all neighboring processes.

Synchronous and asynchronous models are at the two extremes of timing assumptions on the operation of a distributed system. Partially synchronous models assume that bounds on relative timing and operation exist; these bounds may or may not be known to the processes. That is, imprecise knowledge of upper and lower bounds of process execution is available. In practice, processes will often have access to a (imperfect) clock or counter that can be used to infer timing information with regards to the execution and message transmission of other processes.

The distributed system model used here gives us the formalism that is leveraged to design the distributed algorithms used in this dissertation. Several assumptions are used in the overall modeling of the problems in this dissertation, centralized and distributed. The next section discusses these assumptions, the features of these assumptions, and the related consequences.

1.6 Assumptions of this Work

This section discusses the assumptions of this work and how these assumptions affect the expressiveness of the algorithms developed in this dissertation. The focuses of this dissertation are centralized and distributed task assignment and task scheduling. The assumptions that apply to both the centralized and distributed work are as follows.

- The mission is modeled in the form of tasks to be completed and agents to

complete the tasks

- A task's duration is not a function of the agent assigned to complete it
- The capabilities of the agents are binary; for a given task an agent is able to complete it or it is not
- The agents are assumed to perform every task to which they are assigned successfully, incomplete and probabilistic completion are not modeled
- The primary objective is the minimization of mission time. The Tabu/2-opt heuristic is used for this purpose in Chapter III. While this algorithm is implemented for the minimization of mission time here, it is general enough to incorporate other objectives (e.g., total distance, fuel consumed). The distributed scheduling algorithm in Chapter IV is used for mission time minimization; this algorithm is specific to mission time minimization

The centralized part of the work is discussed in the context of vehicle routing. The framework of vehicle routing is primarily concerned with task visitation. The assumptions that apply to the centralized work are as follows.

- Tasks are modeled with zero duration
- The vehicles are assumed to obey unicycle kinematics with no turn-rate restrictions
- Vehicles have a minimum velocity of zero (i.e., are able to loiter)
- Vehicles have a non-zero maximum velocity
- Vehicles are assumed to be unconstrained by capacity or fuel constraints
- Logical (or choice) constraints are not considered; all tasks must be completed

The kinematic and loitering constraints become more realistic as the distance between tasks increases compared to the vehicle turn radius. The effect of the unlimited fuel and capacity assumptions are reduced by 1) our choice of mission time as an objective which inherently minimizes the farthest distance traveled and 2) our choice of the Tabu/2-opt algorithm which, as a secondary objective, minimizes the length of each vehicle's route individually. Precedence constraints are used to describe a class of VRP that is relevant to mission operations.

The distributed work in this dissertation considers agents communicating over a communication network to assign and schedule tasks. The assumptions that apply to the distributed work are as follows.

- The communication network is considered to have a connected topology
- The agents are assumed to use acknowledgement-based communication (e.g., TCP)
- Time is considered as discrete time slots
- Separation of the task assignment and task scheduling problem is used for communication benefit, but may sacrifice optimality.
- When task assignment and task scheduling separation is used the constraints on the task assignment are capability constraints and communication constraints only.

These assumptions are used to incorporate practical features into the task assignment and task scheduling problem. While some of these assumptions simplify the problem, the problems remain difficult to solve and require special tools to overcome this difficulty. The next section discusses this difficulty and common tools used to over come it.

1.7 Challenges and Typical Remedies

Finding an optimal or even feasible task assignment or task schedule is provably difficult. Specifically, for practically useful problems, optimal and constrained task assignment and task scheduling is NP-hard. NP-hardness means that an exhaustive search may be required to find a solution. This search may incur exponential time complexity and the verification of a candidate solution may also incur the same complexity [59]. NP-completeness refers to the possible exponential time complexity of search, but where verification of a candidate solution can be done in polynomial time. The algorithms used to solve problems of this type can be centralized [87, 54] or distributed [83, 19].

We refer to an optimal algorithm as one that is guaranteed to find an optimal solution in finite time. A heuristic does not provide such guarantees, but nonetheless may perform well in practice. Heuristics are often based on intuition about a particular problem. The effectiveness of heuristics can be judged either qualitatively or quantitatively. Heuristics generally come in two types, construction and repair. An algorithm based on a construction heuristic is initialized with a problem instance and builds a solution to that problem instance according to the particular heuristic rule (e.g., expand the nodes of a search tree in a beneficial order). An algorithm based on a repair heuristic is initialized with a problem instance and one or multiple candidate solutions to the problem instance. The repair heuristic then modifies the candidate solution(s) using the heuristic rule until the termination condition of the algorithm is satisfied, at which point the algorithm outputs the best solution found (e.g., vehicle route improvement).

The time-complexity of an algorithm refers to the functional dependence of the time needed to produce an output on the size of the input to that algorithm. If the time needed to produce an output can be upper-bounded by a polynomial function of the input size for all inputs, the algorithm is said to be of polynomial time-complexity.

If the time needed to produce an output can only be bounded above by an exponential function of the input size, the algorithm is said to be of exponential time-complexity. Exponential time-complexity is related to NP-completeness (or NP-hardness) in that the solution of NP-complete (or NP-hard) problems may incur exponential time-complexity in the worst-case.

Optimal task assignment and task scheduling algorithms can be vulnerable to NP-Hardness. For a task assignment and task scheduling problem, the time-scale on which the solution given by an algorithm is executed can be considered. If the time required to produce a solution is larger than the duration of execution, the algorithm may not be suitable for applications where solutions are needed quickly (i.e., online operation). This simple reason often makes optimal algorithms impractical because optimal solutions can simply take too long to produce.

Approximate task assignment and task scheduling algorithms are often designed to have polynomial time-complexity. The requirement to compute a solution faster than its execution places a practical limit on the available algorithms for solving task assignment and task scheduling problems. A heuristic may not reliably give optimal solutions, but in practice the purpose of heuristics is to defeat complexity. Heuristics may be designed to solve entire problems (e.g., vehicle routing heuristics) or to augment existing methods and improve their time-complexity characteristics (e.g., tree search node expansion).

All candidate solutions can be represented by a search-tree [95]. Tree-search techniques such as Backtracking or Branch and Bound [54, 95] can then be used to find a solution that is optimal. When a task assignment and task scheduling problem is solved using a tree-search method, heuristics can offer substantial gains in efficiency. If the number of possible task assignments and the number of possible schedules is finite strong completeness of the tree-search can be guaranteed. That is, the tree-search will return non-existence of feasible solutions. While most heuristics build

or repair solutions to a problem, tree-search heuristics take, as input, a node in a search-tree and output an ordering of that node's children. Tree-search heuristics are able to intelligently expand nodes in a search-tree, thus reducing the time taken to search the tree for a desired solution.

Meta-heuristics such as Genetic Algorithms [53, 88], Simulated Annealing [61], and Tabu Search [40, 38, 82, 65] have also been applied to task assignment and task scheduling problems. These heuristics are similar in the following ways: they are all based on physical principles (i.e., evolution, metallurgy, and hill-climbing); and they are all general in the sense that their original design does not incorporate the specifics of any one optimization problem. These metaheuristics are most useful in a task assignment and task scheduling context when searching in the presence of constraints that restrict the allowable order of task completion and when evaluating solutions with complicated, nonlinear cost functions. These heuristics have the advantage of treating a large class of problems, but provide no guarantee on the solution quality.

The challenges are common to task assignment and task scheduling problems. The problems developed and solved in this work present similar difficulties. The original contributions of this dissertation are in the development and solution of particular problems that address current needs in unmanned air and ground vehicle operations [24]. These problems and the algorithms used to solve them are presented using the modeling framework of this chapter.

1.8 Original Contributions of this Dissertation

The original contributions of this dissertation are as follows:

1. The Tabu/2-opt Heuristic is developed that solves an important Vehicle Routing Problem. The algorithm minimizes mission time subject to precedence constraints. The algorithm finds feasible solutions in fractions of a second and

high-quality solutions to moderate-size problem instances in seconds.

2. A measure of solution quality is developed that uses a stochastic characterization of a problem domain. This quality measure is used to quantitatively compare solutions of combinatorial problem to the space of possible solutions. This gives a relative measure of solution quality for problems where no other meaningful quality measure exists that is feasible to use.

3. The Communication-Constrained Distributed Assignment Problem (CDAP) is developed. The CDAP is important when assigning distributed agents to tasks where direct communication must be guaranteed between agents assigned to constrained tasks.

4. The Stochastic Bidding Algorithm (SBA) that solves the CDAP is developed. The correctness of the SBA is proven. The completeness of the SBA is analyzed, indicating that the SBA finds a solution if one exists. The complexity analysis presents conservative average-case polynomial complexity.

5. The dependence of the number of required communication links on the number of constraints coupling task assignment and task scheduling is quantified. This can be exploited to relax important assumptions regarding the communication topology.

6. The Minimum-time, Arbitrarily-constrained, Distributed Scheduling Problem (MADSP) is formulated. The solution of this problem guarantees the simultaneous satisfaction of all mission constraints and the minimization of mission time.

7. The Optimal Distributed Non-Sequential Backtracking Algorithm (OptDNSB) that solves the MADSP is developed. The OptDNSB Algorithm exploits the structure of the MADSP to guarantee constraint satisfaction and optimality.

The OptDNSB Algorithm is proven correct, complete, and optimal. The complexity of the OptDNSB Algorithm is analyzed; the analysis indicates near-linear average-case complexity for a class of constrained scheduling problems.

8. Conditions are given for which the CDAP and the MADSP can be solved concurrently. The correctness and completeness of the OptDNSB Algorithm running together with an appropriate distributed assignment algorithm are proven. It is shown that the SBA and the OptDNSB satisfy these conditions.

The Tabu/2-opt Heuristic solves the following vehicle routing problem: minimize mission time for several agents to visit a number of waypoints (tasks) subject to precedence constraints relating waypoints and assignment constraints. This heuristic is used to defeat complexity where other methods fail to quickly compute solutions to large problem instances. The Tabu/2-opt heuristic effectively fuses two heuristics; the Tabu Search Heuristic optimizes task assignments; while the 2-opt heuristic optimizes task schedules to minimize mission time. The method is able to produce feasible solutions in fractions of a second and high-quality solutions quickly enough to be practically useful.

The analysis method developed here gives the probability of finding a solution that is better than a candidate solution. The analysis method can be used to compare the cost of a given solution relative to the space of possible solutions. The analysis is based on sampling the space of possible solutions to quantify the statistical distribution of the cost values. Often, no useful analytical lower bound on the optimal solution is available. This method can be used to provide solution quality comparisons when analytical bounds are not available.

The Stochastic Bidding Algorithm is used to solve the Communication-Constrained Distributed Assignment Problem. The CDAP is important in distributed task assignment problems when communication requirements relate agents assigned to constrained pairs of tasks. We consider a network of arbitrary topology. This can limit

the available communication links. The dependence of the number of required communication links on the generality of the problem description allows us to sacrifice problem expressiveness to operate on a network with a reduced number of communication links. This reduced generality is presented as a separation of task assignments and task schedules. The SBA produces a task assignment where the communication requirements needed for distributed task scheduling are satisfied.

The Optimal Distributed Non-Sequential Backtracking Algorithm (OptDNSB) solves the Minimum-time Arbitrarily-constrained Distributed Scheduling Problem (MADSP). The MADSP is the problem of finding a task schedule that satisfies a number of mission constraints while minimizing mission time; knowledge of the tasks, constraints, and the capabilities of the agents are distributed. The OptDNSB Algorithm is correct, complete, and optimal. It is shown here that these properties are maintained while running the OptDNSB Algorithm concurrently with the SBA.

1.9 Dissertation Overview

The rest of the dissertation is organized as follows. Chapter II gives a review of existing literature in the areas of vehicle routing, task assignment and task scheduling, multi-agent systems, and distributed systems. A summary of limitations present in the existing literature is also presented. A brief comparison of the methods developed in the dissertation with existing methods is also given. Chapter III formulates the Vehicle Routing Problem solved in this dissertation. The Tabu/2-opt Algorithm is developed and an analysis of the algorithm's performance is given. Chapter IV formulates the Communication-Constrained Distributed Assignment Problem and the Minimum-time Arbitrarily-constrained Distributed Scheduling Problem. The distributed algorithms used to solve these problems are developed and analyzed. Chapter V gives summarizes conclusions resulting from this work and recalls the original contributions. Appendix **??** presents a list of relevant publications associated with

this work.

CHAPTER II

Literature Review

Consider several vehicles (agents) that move in a two dimensional geographic area, under kinematic constraints, and starting from at least one depot. The mission that the vehicles are to accomplish consists of several tasks. In general the vehicles need not end the mission where they start. The tasks have physical locations within the geographical area and require a certain amount of time to complete. Depending on the problem formulation, the tasks may be referred to as cities or locations; tasks must be "performed" or "visited" by the vehicles.

The mission is specified using constraints that can restrict the order of task completion, the choice of task completion, or both. These constraints are represented generally as predicates that must be satisfied (i.e., evaluate *true*). Given this description of the mission, the agents must find (1) a task assignment specifying which agents do each task and (2) a task schedule specifying the order and choice of task completion such that an objective function is minimized. This objective function typically represents the total distance traveled by the agents or the total time taken to complete the mission.

This problem of task assignment and task scheduling is often considered in a centralized context where the communication between agents is not considered. When the problem formulation incorporates the idea that problem data and the authority to

make decisions are distributed among the agents, the problem is termed distributed. Several bodies of literature including Vehicle Routing, Task Assignment and Task Scheduling, and Multi-Agent and Distributed Systems have addressed aspects of centralized and distributed task assignment and task scheduling problems.

The centralized task assignment and task scheduling work in this dissertation considers kinematically constrained vehicles operating in a two dimensional environment. Precedence constraints are incorporated in the problem description. The objective in this problem is to minimize the mission time. The distributed task assignment and task scheduling work in this dissertation incorporates notions of communication between agents in the form of a communication graph, and distributed authority over the task assignment and task schedule. Constraints on communication, task precedence, and task choice are incorporated. The objective is also to minimize mission time.

This chapter presents a review of related problems and solution techniques. These are relevant to the work of this dissertation in several ways. The Vehicle Routing Problem (VRP) is to dispatch several vehicles to a number of geographically dispersed locations. VRPs often incorporate the following ideas. Only one agent should visit each location, minimization of time required, constraints on which vehicles should visit each location, and precedence constraints restricting the order of visitation. The algorithms that have been developed to solve VRPs have focused on exact and heuristic methods [41]. Several of these ideas are interesting from the perspective of this dissertation and the work on centralized task assignment and task scheduling contributed to this literature.

The Task Assignment and Task Scheduling communities have produced efficient polynomial-time algorithms for task assignment problems involving linear and nonlinear objective functions. These algorithms often have guarantees on the solution quality relative to optimality. That is, they may yield optimality or guarantee solutions

of a certain quality. Scheduling has received much attention from the areas of manufacturing and operations. The scheduling literature addresses problems of scheduling jobs to be performed, where jobs are composed of operations that have precedence relationships between them. Scheduling is provably hard [110]. The requirement that tasks be completed as quickly as possible combined with the constraints on order of completion have motivated efficient algorithms to solve large-scale scheduling problems. The Task Assignment and Task Scheduling literature provide direct motivation and insight for the work of this dissertation.

The Multi-Agent Community has developed frameworks for cooperative multi-agent groups to collectively solve problems. This community has addressed such fundamental issues as quantifying the coupling inherent in a system of agents whose actions can affect each other. Various general formulations for multi-agent planning such as Distributed Constraint Satisfaction and Distributed Optimization have come from this community. This community gives insight to the nature of distributed problem solving, which has helped in the design of the formulations and algorithms explored in this dissertation.

The Distributed Systems Community has traditionally addressed problems of distributed computing. Fundamental notions concerning distributed agreement, fault detection, and definitions of time for distributed systems have originated in this field. The ideas of synchronous and asynchronous timing assumptions and the associated consequences for distributed computing began in the distributed systems community [37, 67]. Algorithms for leader election, distributed agreement, distributed optimal network construction, and failure detectors have been designed by this community [75, 16, 69]. This work is referenced to provide overall context to the discussion of distributed task assignment and task scheduling. Some of the models developed for distributed systems are used herein.

2.1 The Vehicle Routing Problem

The Vehicle Routing Problem (VRP) originated in truck dispatching [25]. The problem is: given several vehicles and several locations for them to visit; find routes for all of the vehicles such that the total length of all routes is minimal. This problem plays a critical role across several applications including truck dispatching, supply chain management, and organizational optimization [43, 41]. Several variants of this problem consider constraints on vehicle carrying capacity, precedence constraints, and different objective functions. The discussion of this section considers agents as vehicles.

It is common to use a graph-theoretic representation for the VRP; however, a task assignment and task scheduling framework is used here for its ability to incorporate explicit notions of time. For instance, in Chapter III precedence constraints are used to restrict the order of task completion; therein it is not important that the same agent perform constrained tasks, only that the timing constraints be obeyed. The VRP can be stated generally within the task assignment and task scheduling framework as

$$\min_{TA,TS} J(TA, TS), \tag{2.1}$$

$$\text{s.t. } p(TA, TS). \tag{2.2}$$

That is, given an objective and constraints that are functions of the assignment of vehicles to tasks and the order of task visitation, find a task assignment and task schedule that minimizes the objective $J(TA, TS)$ while satisfying the constraints $p(TA, TS)$.

The objective functions used here are discussed in terms of the length of vehicle routes. For VRPs of the following sections, this can be expressed as time or distance. The time an agent $a_i \in \mathcal{A}$ spends to visit the tasks assigned to it is a function of the task schedule and task assignment and is $T_{ei}(TA, TS)$. For several of the problems

discussed below decision variables can be chosen for the problem such that T_{ei} is a linear function of the decision variables; for the problem of Chapter III, it is not. This section presents a review of the Vehicle Routing Problem literature including a number of prominent variants of the problem.

2.1.1 The Classic Traveling Salesman Problem

The Traveling Salesman Problem (TSP) can be stated as follows,

$$\min_{TS} T_{e1}(TS), \text{ s.t.} \tag{2.3}$$

$$\frac{d(t_i, t_j)}{v(1)} \leq |TS(t_i) - TS(t_j)|, \tag{2.4}$$

where for the TSP, $TA(t_i) = TA(t_j)$, $t_i \in \mathcal{T}$, $t_j \in \mathcal{T}$. Equation 2.4 reflects the minimum travel time between two tasks, where $d(t_i, t_j)$ is the distance between tasks t_i and t_j, and $v(1)$ is the unit velocity. The objective is to minimize the time to visit all tasks. This model assumes vehicles with unit velocity and first order unicycle kinematics.

The TSP is the canonical VRP. The search for better solution methods has been the source of much advancement in combinatorial optimization. When solved to optimality, the problem size can scale exponentially with the number of locations to be visited. There are heuristics available to solve the TSP approximately. A survey of these is given in [1].

Construction heuristics given in [1] for the TSP include the Nearest Neighbor, Greedy, Clarke-Wright, and Christofides heuristics. The Nearest Neighbor heuristic builds a TSP tour by iteratively adding unvisited cities to the tour. The salesman begins at a starting city and then visits the next closest unvisited city. The Greedy method is based on a graph theoretic interpretation. Consider a graph with the cities as vertices and an edge between each pair of vertices with length equal to the distance

between the two vertices. The Greedy heuristic begins by selecting the edge with the smallest distance. The available edge with the shortest length is added to the tour; this is repeated to form a complete tour. The edges added must never cause a vertex in the tour to have degree more than two.

The Clarke-Wright Heuristic begins by choosing a single city as a hub. The tour is iteratively built by replacing edges to the hub by edges between cities that shorten the tour. The Christofides Heuristic [21] provides the best quality of any existing TSP construction heuristic, giving tours within $\frac{3}{2}$ of optimal for any instance. This heuristic first constructs a minimum spanning tree and then a minimum-length matching is built using the odd vertices of the spanning tree. The matching is combined with the spanning tree to yield a new graph. An Euler cycle [21] is then extracted from this graph.

The version of the 2-opt move used here, and its higher order variants, are the basis for many TSP heuristics including the use of Tabu Search heuristics [62], Simulated Annealing [9], and Genetic Algorithms [91]. The most successful heuristic is a variation of the original Lin-Kernigan (LK) TSP Heuristic [73]. The LK Heuristic is a repair heuristic that accepts as input an initial solution to the TSP and modifies the solution by changing the order in which cities are visited. The most effective implementation of the LK Heuristic is that of [44, 45], the LKH Heuristic. The LKH Heuristic is able to find optimal solutions for all standard test instances of the TSP, holds the record for the largest TSP solved to-date, and has an average-case time complexity of $\mathcal{O}(n^{2.2})$, where n is the number of cities. This implementation of the Lin-Kernighan heuristic holds the record on solution cost for all TSP problem instances in the TSPLIB, a collection of standard test instances for the TSP.

2.1.2 The Precedence Constrained Traveling Salesman Problem

For the Precedence Constrained Traveling Salesman Problem (PCTSP), we define a set $\mathcal{P}_i$ (possibly empty) for each task $t_i \in \mathcal{T}$ as the set of tasks that must precede t_i in the schedule. The PCTSP can be stated as follows,

$$\min_{TS} T_{e1}(TS), \text{ s.t.} \tag{2.5}$$

$$\frac{d(t_i, t_j)}{v(1)} \leq |TS(t_i) - TS(t_j)|, \tag{2.6}$$

$$TS(t_i) > TS(t_j), \forall t_j \in \mathcal{P}_i, \tag{2.7}$$

where for the PCTSP, $TA(t_i) = TA(t_j)$, $t_i \in \mathcal{T}$, $t_j \in \mathcal{T}$.

The PCTSP is treated in [3] and is concerned with satisfying precedence constraints dictating the ordering of certain tasks. Here, the objective is to find a feasible tour, satisfying all precedence constraints, with minimal cost. The method of [3] uses a cutting plane algorithm to exclude portions of the search space that violate precedence constraints. The procedure is an iterated linear programming scheme. When a precedence constraint violation is detected, the linear program is augmented with a new constraint, or cutting plane, and continues. The method can suffer from a possible exponential increase in the number of additional constraints, but in practice can solve instances of several hundred cities.

The work of [102] provided a Branch and Cut formulation that results in an improvement in the worst-case number of constraints to polynomial. This work addresses symmetric and asymmetric TSP formulations. More recently much work has been done to consider precedence constraints in the context of pick-up and delivery problems [30, 23, 22] (Section 2.1.4).

2.1.3 The Multiple Traveling Salesman Problem

The Multiple Traveling Salesman Problem (mTSP) can be stated as follows,

$$\min_{TA,TS} \sum_{a_i \in \mathcal{A}} T_{ei}(TA, TS), \text{ s.t.} \tag{2.8}$$

$$\frac{d(t_i, t_j)}{v(1)} \leq |TS(t_i) - TS(t_j)|, TA(t_i) = TA(t_j), \tag{2.9}$$

where $t_i \in \mathcal{T}$, $t_j \in \mathcal{T}$.

The problem is: given several vehicles and several locations for them to visit; find routes for all of the vehicles such that the total length of all routes is minimal. This problem is been addressed in the literature through integer linear programming formulations [55, 5] and through the use of transformations to the single TSP [42].

A survey of the mTSP and various formulations is given in [5]. This work presents variations on the problem including time windows, fixed costs for including additional vehicles, and lower and upper limits on locations visited by a vehicle. Applications include scheduling of personnel, routing of school buses, scheduling of manufacturing operations, and mission planning.

The integer linear programming formulation of the mTSP given in [55] presents improved subtour elimination constraints, the constraints restricting a single vehicle to one and only one tour. The primary innovation of this work is to include a minimum and maximum number of locations to be visited by each vehicle. This is a generalization that can be used in practice to require specific distribution of effort among the vehicles. The formulation of [55] is able to provide computational speed-up for problems of moderate size.

2.1.4 Pick-up and Delivery

The Pickup and Delivery problem is a variant of the VRP where the vehicles must pick-up packages and are constrained to deliver them after pick-up. This variant includes elements from the PCVRP and the mTSP. A basic variant of the Pick-up and Delivery problem is stated as follows,

$$\min_{TA,TS} \sum_{a_i \in \mathcal{A}} T_{ei}(TA, TS), \text{ s.t.} \tag{2.10}$$

$$\frac{d(t_i, t_j)}{v(1)} \leq |TS(t_i) - TS(t_j)|, TA(t_i) = TA(t_j), \tag{2.11}$$

$$TS(t_i) > TS(t_j), \forall t_j \in \mathcal{P}_i, \tag{2.12}$$

$$TA(t_i) = TA(t_j), \forall t_j \in \mathcal{P}_i, \tag{2.13}$$

where the constraint of (2.13) physically means that the same vehicle must pick-up and deliver a package.

In [30] an analysis for the single TSP with pick-up and delivery is given. This formulation is based on integer linear programming and an analysis is given of the polyhedral structure of the problem. The proposed Branch and Cut algorithm is able to solve problems with an improved number of precedence constraints over previous methods.

In [23] a similar pick-up and delivery problem is addressed. Here, the additional constraint is included to ensure that deliveries are done following a last-in first-out policy. Inequalities are proposed that allow the formulation of this problem as an integer linear program. The problem is solved by a Branch-and-Cut Algorithm. The first-in first-out version of this problem is treated in [22]. The problem of pick-up and delivery, without the LIFO or FIFO restrictions, with time windows is formulated and solved in [94]. This work introduced the Branch and Cut and Price algorithm

that uses improved lower bounds for increased efficiency over previously used Branch and Cut algorithms.

The work of [84] treated a version of this problem. This work provided a Tabu Search-based heuristic for this problem. The features of the problem formulation include multiple vehicles, a single vehicle depot, vehicle capacity limits, limits on vehicle travel distance, pick-up and delivery constraints, and a linear (i.e., sum of distances) objective function. This work shows that a Tabu Search-based heuristic can be used to find high-quality solutions to VRPs with a variety of constraints.

In [12] the Fleet Size and Mix Vehicle Routing Problem is formulated and solved using a Tabu Search Algorithm. This formulation includes vehicles of different capabilities, capacity constraints, a fixed cost for using vehicles of different types, and a variable travel cost. The objective is to minimize the sum-total cost of servicing customer demands. This heuristic first constructs routes using a nearest neighbor heuristic. The Tabu Search moves customers between agents' routes to improve cost.

Another Tabu Search Algorithm is applied to the VRP with simultaneous pick-up and delivery in [114]. This algorithm overcomes the scalability problems of [85]. The work of [85] is based on an Integer Linear Programming formulation and is only able to handle small problem instances.

2.1.5 Unmanned Air Vehicle Mission Planning

Modern mission planning is taking advantage of constrained task assignment and task scheduling formulations. A survey of cooperative decision and control methods applied to unmanned air vehicles (UAVs) is given in [104]. This work motivates the use and coordination of several vehicles to accomplish a military mission. The motivation includes the ability to collect and share information using distributed agents, effective management of resources, and robustness to failures. Further motivation for these abilities are provided directly from the U.S. Air Force in their annual Technology

Report, [24]. This report details the Air Force's vision for the development of their operational capabilities over the next twenty years.

A Capacitated Transshipment Assignment Problem (CTAP) formulation is used for Unmanned Air Vehicle (UAV) task assignment in [104]. These methods are not computationally intensive and are scalable, but involve very little scheduling of the agent routes. This formulation is designed to model the real-time assignment of tasks to agents. Agents are iteratively assigned one task each; tasks are assigned as agents complete them. This process is repeated until all tasks have been assigned.

Mixed Integer Linear Programming (MILP) formulations are presented in [104] that describe missions with precedence constraints (between distinct routes, fuel constraints, and minimization of mission time. Tree Search formulations have also been used to the same effect. However, in practice the solution time for both methods scales exponentially with the number of tasks and agents. In practice, these tools are only viable for small problem instances.

The tools of distributed consensus (see Section 2.3.4) are used in [104] to develop distributed estimation algorithms. This work develops a distributed information filter used to estimate the positions of distributed targets and agents. This is a means by which each agent could estimate a global picture of the environment using corrupted and latent local information from the other agents.

The modeling of adversarial missions is done in [33]. In this work, the adversaries have the ability to cooperate and improve their attacking capabilities to achieve better results. With this knowledge, the UAVs must formulate a mission plan to capture a high-value asset from adversarial agents. This work allows the explicit trade-off of risk and reward in mission design. The authors detail UAV adversarial missions further in [32]. This work includes UAVs with heterogeneous capabilities and Linear/Metric Temporal Logic mission specifications.

The work of [56] develops Linear Temporal Logic (LTL) tools that can express

a wide variety of UAV missions. These tools are used to generate a set of MILP constraints. The resulting optimization problem is solved in the MILP framework using standard optimization tools. The novelty of this approach is the application of LTL mission specifications to UAV mission planning and the constraint generation algorithms. Along the same thread of research, including Metric Temporal Logic constraints in a VRP is done in [57]. This problem formulation allowed for time-windows in the pick-up and delivery problem. This is in addition to the mix of temporal and logical constraints.

Decentralized perimeter surveillance is studied in [60]. The problem of having multiple agents guarantee coverage of a possibly changing perimeter is addressed. The goal here is to gather and relay information from the perimeter to a central location. The agents must ensure convergence in the presence of communication range limitations. The algorithm guarantees that the agents converge to a patrol pattern that assigns equally portions of the perimeter to the agents. The paper presents solutions for agents with and without kinematic constraints.

Cooperative search scenarios like that of [123] are important when a group of autonomous agents are tasked with the pursuit and capture of a moving target. The work of [123] assumes a target with probabilistic behavior with known distribution. The goal is for the autonomous agents to use knowledge of the target's likely motions to cooperatively capture the target. This problem considers urban terrain where the presence of buildings can occlude vision. The agents also have different capabilities, i.e., the unmanned ground vehicles are slow, but can capture the target and the unmanned air vehicles are fast, but cannot capture the target. An auction method is used to solve the problem.

Near-optimal coalition formation for the purpose of cooperative target prosecution can allow unmanned aircraft to pool limited resources in an attack [77]. In [77], the problem is: given a set of UAVs, find an optimal subset of UAVs to be assigned

to prosecute a target in minimum time. A Particle Swarm Optimization technique is used to overcome computational complexity. Their method gives near optimal solutions to this problem.

The work of [51] and [50] developed a novel heuristic for a precedence constrained mission planning problem. The problem formulation incorporates vehicle velocity (lower and upper) bounds, precedence constraints, multiple vehicle depots, distinct vehicle dispatch and recovery locations, and minimization of mission time.

The centralized work of this dissertation is presented using a vehicle routing framework. The problem formulation expresses precedence constraints that require agent timing coordination, agent coordination for cooperative task completion, capability constraints, and minimization of mission time. This VRP framework is able to express multi-vehicle operations where the vehicles must be routed between locations in the Euclidean plane. The Tabu/2-opt heuristic used to solve this problem is able to satisfy these constraints and minimize mission time. Task assignment and task scheduling are used to frame the mathematics through out this dissertation, centralized and distributed. Additionally, task assignment and task scheduling tools are used prominently in the development of the distributed work. As such the task assignment and task scheduling literature is reviewed here.

2.2 Task Assignment and Task Scheduling

Task assignment and task scheduling are used by agents to determine who will do which tasks and when those tasks will be performed. This section presents a review of the task assignment and task scheduling literature. This section also discusses distributed task assignment and distributed task scheduling.

2.2.1 The Task Assignment Problem

The assignment problem is to assign N_t tasks (or objects) to N_a agents such that some utility (or objective) function is maximized (or minimized). The utility function couples the assignments of the agents. Maximizing the utility of a task assignment can be done in a centralized or distributed way.

A detailed survey of task assignment formulations is given by [39]. This survey discusses task assignment formulations and algorithms in the context of objective function minimization. The classic auction algorithm which minimizes a linear objective function subject to capability constraints is discussed. This work formalized the notion of agents that have the capability to perform several tasks, the notion of tasks that require several agents to be performed, and the notion of instantaneous versus time-extended assignments. The latter notions deal with the ability to make task assignments that consider the future state of the world. The complexity and optimality of various methods is investigated within this context.

The Hungarian Algorithm for the assignment problem solves a linear unconstrained assignment problem [64]. The Hungarian Algorithm is the first polynomial-time algorithm used to solve the assignment problem and find an optimal solution. Prior to [64], the Hungarian Algorithm is used to assign workers to jobs considering only whether they were qualified or not. This binary measure of suitability is extended in [64] to a real measure of benefit. The Hungarian Algorithm for the assignment problem has complexity $\mathcal{O}(N_a N_t^2)$.

A modern application for assignment algorithms is that of robot soccer [108, 115, 112]. Robot soccer is a dynamic, multi-agent environment where the information needed by the robots to assign tasks changes dynamically. This assignment problem contains single-agent tasks (i.e., task requiring only one agent) and single-task agents (i.e., agents capable of performing only one task). In this setting, the task assignment must be computed on the order of milliseconds. These algorithms must also address

the problem of partially-synchronous communication of required information. For this real-time application, greedy assignment algorithms perform well.

The assignment of multiple agents to a task can be done using coalition formation techniques [100]. This work addresses the problem of combinatorial optimization by self-interested distributed agents. The agents are solely interested in maximizing their own benefit. However, several of the tasks are better achieved through cooperation. This work explicitly incorporates computational limits into the problem formulation. This model allows agents to cooperate to overcome high computation costs. It also explicitly excludes the possibility of computing optimal solutions for large problem instances. The coalition formation methods of [100] are applicable across many problems, but are mentioned here for the applicability to task assignment.

In [28] a leader-based approach to task assignment is used. This approach is a market mechanism where agents bid for tasks selfishly, but may propose solutions involving other agents. If these local solutions provide more benefit than selfish assignments, the solution is adopted by the agents. Because the assignment problem is combinatorial, there is an upper limit on what a single agent can compute. However, evaluating a group of assignments rather than individual assignments can improve the quality of the assignments.

Negotiation is a paradigm that is used to frame task assignment problems. Negotiation can be used by agents to determine a task assignment. Auction algorithms are tools that agents can use for effective negotiation. The following section discusses negotiation and auctions in this context.

2.2.2 Negotiation and Auctions

Negotiation is a process by which a joint decision is reached by two or more agents, each trying to reach an individual goal or objective [118]. An auction is a market institution with an explicit set of rules determining resource allocation and prices on

the basis of bids from market participants [80].

Negotiations and auctions span the topics of economic interactions, multi-agent planning, scheduling of resource usage, assignment problems, and vehicle routing. The work of [80] examined auctions as a device for the exchange of information between buyers and sellers. This work studied price equilibria and the equivalence of the English, Dutch,First-Price Sealed-Bid, and Vickery Auctions. The work of [80] studied the effects of asymmetric information, incentives for bidders, and bid correlation on the optimality of these four types of auctions.

The classic auction algorithm of [7] is used for distributed resource allocation problems with linear objective functions. This algorithm solves the problem of assigning N_t tasks to N_a agents where $N_t = N_a$. The agents bid in a greedy way, that is, each agent places a bid for the task that is the best value. The value of a task for an agent is defined as the benefit of that agent being assigned the task minus the price paid for the task. The prices are raised incrementally until the exchange of tasks reaches an equilibrium. The classic auction algorithm of [7] shows that the classic assignment problem can be solved in linear time. That is, the classic auction algorithm has a time-complexity linear in the number of tasks being assigned. The only constraints that the classic auction algorithm can cope with are those associated with the capabilities of the agents. This type of constraint is dealt with by restricting the set of agents that may bid on a task.

The assignment problem of [7] can be stated in terms of agents and tasks as follows. Consider the set of $N_a > 0$ agents in (3.2) and the set of $N_t > 0$ tasks in (3.1). Let b_{ij} be the benefit of assigning task t_i to agent a_j. The assignment problem is to find an assignment TA of tasks to agents such that

$$J(TA) = \sum_{(t_i,a_j)\in TA} b_{ij} \tag{2.14}$$

is maximum.

The basic low complexity auction algorithm of [7] is as follows. Let i_j satisfy $a_j = TA(t_{i_j})$. The auction algorithm of [7] is a repair heuristic that operates on the assumption that each task t_i has a price p_i that an agent must pay in order to be assigned to it. The algorithm seeks an assignment such that,

$$b_{i_j j} - p_{i_j} = max_i\{b_{ij} - p_i\} - \epsilon. \tag{2.15}$$

where ϵ is called the slack variable. The market mechanism used to achieve this requires that each agent a_j place a bid for a task t_{i_j} where

$$i_j \in \arg\max_{i:t_i \in \mathcal{T}}\{b_{ij} - p_i\}. \tag{2.16}$$

Each unassigned agent a_j places a bid $\gamma_j = v_j - w_j + \epsilon$ where

$$v_j = max_i\{b_{ij} - p_i\}, \tag{2.17}$$

$$w_j = max_{i,i\neq i_j}\{b_{ij} - p_i\}, \tag{2.18}$$

and the price p_i is raised by γ_{j_i} upon agent a_{j_i} winning task t_i. This algorithm repeats this process until $b_{i_j j} - p_{i_j} \geq max_i\{b_{ij} - p_i\} - \epsilon$ for all tasks $t_i \in \mathcal{T}$. The basic auction algorithm terminates in $\mathcal{O}(\frac{\max_{i,j}|b_{ij}|}{\epsilon})$ iterations and with $\epsilon < \frac{1}{N}$ the algorithm terminates with an optimal assignment.

The work of [101] detailed mechanisms for negotiating agents to assign resources amongst themselves. This work provided a game theoretic and a distributed systems point of view. Auction protocols, contracting, coalition formation, and deceitful agents are addressed in the Sandholm dissertation.

The work in [29] presents a method for resource allocation among distributed agents where the agents' preferences are induced by a Markov Decision Process

(MDP). A critical assumption here is that the agents' transitions between states of the MDP and the rewards received are independent of each other once the resources are allocated. Additionally, resources may not be reallocated once allocated. The objective function being minimized is quasi-linear and the agents bid on bundles of resources. The enumeration of possible bundles can result in an exponential increase in complexity with respect to the cardinality of the set of resources. A MILP formalism is used to address this issue. A central auctioneer is required to moderate the auction.

The work of [116] treats a scheduling problem where time slots are assigned to agents who must schedule the execution of jobs. The scheduling problem is posed as an exchange economy and is treated using market-oriented programming (i.e., auction mechanisms). This work discusses fundamental factors that effect the existence of equilibrium solutions and gives conditions for such existence. Several auction mechanisms and the associated convergence, complexity, and optimality characteristics are discussed.

In the negotiation and auction work of [101, 29, 116], privacy of information is a concern. Privacy of agent information is not an explicit concern in the current dissertation. This difference is due to the separate communities these bodies of work serve. The current work considers agents that are seeking to maximize a global benefit. Bidding is used as a tool to accomplish this. The benefits of auction mechanisms exploited here are their distributed operation and efficiency.

The classic auction algorithm is extended by [19]. The auction algorithms of [19] provide the following extensions to the classic auction algorithm. Data aggregation is not necessary; a large class of nonlinear objective functions is accommodated; bounds on the distance of cost value from optimality were given; and $N_t \geq N_a$ is allowed. The extension to a distributed framework is done by having agent relay bids across the network and use conflict resolution to determine winners. A similar technique is used

here. The extension to nonlinear objective functions is done for a class of objective functions where the function value increases monotonically with the assignment of multiple tasks to a single agent. Optimality bounds are given by showing that the algorithm performs no worst than a greedy algorithm.

The Contract Net Protocol (CNP) of [105] is a task sharing protocol. It is designed to allow agents to enlist their peers to assist with performing tasks that can better be performed with or by other agents. The CNP operates as follows. An agent responsible for a task can choose to ask other agents, that it may communicate with, to share in performing the task or accept responsibility for the task altogether. In the single task case, an agent (the contracting agent) first sends a request for bids. The request is sent to the agent's neighbors. If the other agents choose to bid for the job, they will send their bids. If the bid values are acceptable, the contracting agent grants the contract to perform the task to the highest bidder. For a task that can be separated into sub-tasks to enlist the help of others, the same procedure is repeated for the sub-tasks. The work of [26] extends this contracting behavior into a negotiation paradigm that can be used to formalize distributed problem solving more broadly. The authors focus on using the idea of negotiation as a cooperative tool. The CNP is used to implement this tool. The CNP is used for vehicle routing in [11]. The CNP optimizes the assignments of locations to vehicles, while an insertion heuristic optimizes the routes of the vehicles.

In [52] the problem of Communication-Constrained Distributed Assignment (CDAP) is formulated and solved. The CDAP is the problem of assigning tasks to agents such that agents assigned tasks sharing a constraint can communicate directly. This work presents a correct randomized algorithm, the Stochastic Bidding Algorithm (SBA), to solve the CDAP. For an algorithm to solve the CDAP, it must find an area of the communication network that contains the required subgraph. This problem is a cousin of the graph matching problem which is NP-complete. The SBA is shown to

find a solution if one exists and to have linear average-case complexity.

The Stochastic Bidding Algorithm developed in Chapter IV is used for constrained task assignment. It is similar to the above auction algorithms in that it is used by agents to bid for tasks. Unlike the above techniques, this algorithm solves a nonlinear task assignment problem using a distributed stochastic search. In our discussion of task assignment and task scheduling task scheduling is used to determine the times at which tasks will start and finish. The following sections reviews the task scheduling literature.

2.2.3 The Job-Shop Scheduling Problem

Scheduling has often been considered separately in the form of the Job-shop Scheduling Problem (JSP) that originated in manufacturing. In many practical cases scheduling is NP-Hard. The NP-completeness of two fundamental classes of JSPs is proven in [110]. In [110] it is shown that the JSP with two machines, each capable of performing all tasks, and the JSP where all operation durations are equal, are both NP-complete. This result is important because it verifies the NP-completeness of problems that are generalizations of these two. The analysis of [110] considers the problem of verifying the existence of a schedule given a known fixed makespan. When minimization of the makespan is considered, the problem of verifying the solution becomes as difficult as finding the solution, and is NP-hard.

The work of [17] and [18] review Genetic Algorithm (GA) methods for solving JSPs. When using a GA, it is important to describe the problem such that it is solvable by a GA. This description is the solution representation. In [17] various types of solution representations are reviewed in detail. The intention is to discuss the ability of each representation to cope with the constraints of the JSP. In [18] the various crossover operations associated with each type of solution representation are reviewed in detail. A GA is presented in [53] that solves the problem of scheduling

jobs amongst several factories, each with various capabilities. The factories in this work are considered to be geographically distributed, but the solution method is centralized. The solution method is an adaptation of a centralized Genetic Algorithm-based method for the problem of [53]. A GA is introduced to solve the flexible JSP (FJSP) in [88]. The FJSP is a generalization of the JSP where, for each job, a machine must be chosen to perform the job from a set of capable machines. That is, in the FJSP, the assignments of operations to machines has not been done *a priori*.

In [98] and [97] a number of heuristics are evaluated that can effectively reduce the complexity of Backtracking searches. Specialized heuristics for the JSP are developed that are able to use constraint satisfaction methods to minimize makespan. These heuristics are applied to the JSP problem domain for this analysis. Several heuristic improvements for Backtracking searches are detailed in [119]. These heuristics adjust the order in which a Backtracking search should proceed to assign times to tasks so as to avoid backtracking. Several of these heuristics are also given in [95].

The work of [74] addresses the problem of job scheduling using a due-date-based heuristic. This heuristic incorporates the precedence constraints of the JSP. The objective function used is weighted tardiness. This objective function is used to penalize the sum of the lateness of completion of the jobs being scheduled. It is a linear function of the finish times of the jobs and the weights must be determined *a priori*. This work is based on Dispatch Scheduling, a greedy scheduling procedure that is used for real-time scheduling and execution.

The Shifting Bottleneck Procedure is described in [2]. The Shifting Bottleneck Procedure is a construction heuristic that gives priority to the machine that is the source of the bottleneck. That is, the heuristic attempts to build a schedule by scheduling the machine that is the source of the bottleneck first while locally optimizing the individual schedules of the other machines.

Simulated Annealing can be used to solve the JSP [113, 72]. The work of [113]

presents a formulation that gives good solutions to the JSP where the objective is to minimize makespan. The asymptotic convergence to optimality is proven. The work of [63] presented a method to counteract the high computational-time cost of using Simulated Annealing to solve the JSP. The problem presented in [63] treats a typical JSP with a unique, distributed method of computation. In this work the authors were able to distribute the computational effort of solving the JSP over several computational nodes.

2.2.4 Distributed Scheduling

Distributed scheduling concerns the scheduling of tasks when the knowledge of the tasks, scheduling constraints, and (possibly) knowledge of the schedule are distributed.

Scheduling computational jobs on computer clusters is addressed in [124]. This work addresses real-time task scheduling and optimizes the associated quality of service (QoS). Additional concerns of this work are fault-tolerance. The algorithm developed for this scheduling problem is capable of functioning when cluster computers fail. This is accomplished by maintaining backup copies of jobs and scheduling the backup jobs at later times than the respective primary job. This scheduling is adjusted to maximize quality of service.

In [27] a distributed auction formulation is developed for a JSP including task assignment and capacity constraints. This novel approach applies Lagrangian Relaxation to incorporate scheduling constraints into a cost minimization procedure. Two complementary auctions are used. The first implements an auction between agents to minimize a price paid by each agent for tasks it is assigned. The second implements an auction between tasks to minimize resources used as a function of the Lagrange multipliers. This method is able to simultaneously minimize a linear cost function and determine the Lagrange Multipliers that augment the cost function with the

constraints. This work assumes a complete communication graph.

One important approach to distributed scheduling is via the Temporal Decoupling Problem (TDP) [47, 46, 90]. The TDP can be described generally, but is described in the task scheduling framework as follows. We are given several tasks that must be scheduled by distinct agents where the scheduled times of the tasks must obey a set of precedence constraints. We are also given a partitioning of the tasks, each agent corresponding to a subset in the partition. The goal is to find the following: a set of constraints corresponding to each agent; such that each agent may independently schedule its tasks subject only to its set of constraints, while guaranteeing satisfaction of the original timing constraints. This is subject to the existence of a solution. The time points together with the original precedence constraints are called a Simple Temporal Network (STN).

A family of sound and complete algorithms is given in [47] for solving the TDP. The topics of augmenting, controllability of, and distributing control of STNs is covered in [48]. Augmented STNs contain a time point that represents the current time. The aim of distributing control of an augmented STN to several agents is for the agents to be able to perform their tasks considering the decoupled STNs without interfering with each other.

Optimal Temporal Decoupling is addressed in [90]. The OTD Problem is to find a solution to the TDP that maximizes some metric h that is a function of the decoupling [46]. The work of [90] demonstrated that the OTD Problem is NP-hard. This work also offered classes of the metric h that facilitated the polynomial-time solution of the OTD Problem using Linear Programming. In [89] algorithms are offered that improve on the previously best known time and space complexity of checking the consistency of Simple Temporal Networks (STNs). That is, they provide efficient algorithms for checking to be sure a solution exists that satisfies temporal (or precedence) constraints.

The problem of distributed scheduling subject to uncertain completion times is addressed in [106]. Uncertainty is addressed in two ways. First, the idea of explicitly inserting slack into the schedule between constrained tasked is used to absorb unexpected delays in execution. This is done using a STN problem formulation. Second, rescheduling is used to account for unexpected changes that will cause constraint violations.

The work of [49] addresses arbitrarily-constrained distributed scheduling. This method involves using a discrete-time representation and incorporating temporal and logical mission constraints using a constraint satisfaction problem framework. Representation of the scheduling problem as a DCSP is used here to solve the MADSP. This work is extended in this dissertation to the OptDNSB Algorithm that gives minimum-time schedules. The scheduling goals of this dissertation are twofold. The first is the inclusion of constraints that can express logical and temporal restrictions of mission planning. The second is the minimization of mission time. This is accomplished in Chapter IV.

2.3 Multi-Agent and Distributed Systems

Multi-agent systems [118, 31, 10, 111, 26] is a broad field that studies the interaction of agents (often computational) that interact for the purpose of achieving cooperative or selfish interests, or both. This section reviews work in this field.

2.3.1 Distributed Problem Solving

Distributed Problem Solving is largely concerned with the study of cooperative agents. This field addresses the challenges of having distributed agents cooperate to solve problems given that their actions, goals, and computing abilities are distributed.

An exposition is given in [118] covering topics such as multi-agent decision making, problem solving, search, learning, practical applications, etc. This work intro-

duces agents as being able to perceive their environment, respond to changes, exhibit goal-directed behavior, and interact with other agents. Agents use these abilities to accomplish their own, and possibly global goals. In [118] the topics of agent communication (languages), interaction protocols are addressed; planning representations, task sharing and coordination protocols are given. Multi-agent constraint satisfaction, optimization, and rational decision making considering selfish and dishonest agents are discussed. Multi-agent learning, organization, and reasoning formalisms are given. Additionally, [118] discusses practical agent-based programming and software design.

Early work on distributed problem solving can be found in [71, 31]. The work of [71] developed the idea that the distributed system need not provide a correct output at all times. The idea is that the cooperating agents could iteratively solve pieces of a global plan and that this plan can eventually be satisfactory (by some measure). Additionally, the agents can potentially reduce communication requirements by computing partial results from data collected locally and then communicate more distilled information. The idea here is to enhance robustness, reduce communication costs, distribute processing, and increase the ability of the system to respond quickly to new data.

In [31] partial global planning is considered as dynamic coordination between agents. It is some of the first work to provide a general framework for distributed agents to design consistent global plans using local information and coordination with neighbors. Partial Global Planning is a technique for distributed problem solving where each of the distributed agents produces a local plan that addresses its needs. Agents use plan information from their neighbors to adjust their own plans to avoid conflicts. The idea is for agents to plan locally to accomplish their local goals, then to adapt those plans through communication with neighbors to achieve coordination.

The work of [10] considers the fundamental coupling relationships between the agents' own actions and a group of cooperating agents. This work also explores

the coupling effect of the particular problem the agents are solving. This work is fundamental in that it addresses the concept of "distributed-ness" as opposed to a multi-agent system being distributed or centralized. This work is able to quantify coupling in terms of two parameters, one dependent on the coupling inherent between the actions of the agents, and the other dependent on the particular problem being solved.

In [86] the concepts of [10] were explored through implementation of a Distributed Constraint Satisfaction Problem planning algorithm. Their algorithm used heuristic value ordering combined with local search strategies for pruning to achieve efficiencies in the CSP search. The search also gives preference to actions that are judged more likely, by the search tool, to yield goal-achieving plans. In essence, these tools help to search the solution space in a more effective order and quickly prune unfruitful portions of the solution space. Distributed Constraint Satisfaction is reviewed in further detail in Section 2.3.2.

Agents can coordinate their actions with others if they know the effects of those actions. These effects must be modeled; one approach is to model these effects using learning algorithms [125]. In this work, the actions and the constraints of the environment are known. A satisfaction scheme uses this knowledge to generate models for the actions that can then be used for multi-agent planning. In principle, this method allows the designer to focus on specifying the actions and the constraints rather than manual model design.

Another way to consider multi-agent planning is using multi-agent Markov Decision Process tools [6, 107]. This type of framework uses a model of the environment that is discretized into states, whereby agents may probabilistically transition between states by employing actions. When in a state an agent receives an observation and a reward. The observation and reward functions may differ between agents. The aim is to find a policy (or plan) for each agent; this plan dictates its actions for the

state it believes it is in. This policy will determine how the agent will interact with its environment and with the other agents. The work of [107] contributes a toolbox for modeling and testing multi-agent planning systems using Markov Decision Process frameworks.

2.3.2 Distributed Constraint Satisfaction and Optimization

A review of constraint satisfaction problem (CSP) solution techniques is given in [4]. This review discusses formulation of planning problems as CSPs. Techniques for solving CSPs are discussed and speedup techniques based on heuristics are presented. Scheduling problems are also discussed in [4] and heuristics for specific problems are given. The link between planning and scheduling is illustrated. That is, planning is presented as the problem of determining which tasks to perform and scheduling as the problem of deciding when to perform the tasks.

The work of [121] studied the Distributed Constraint Satisfaction Problem (DCSP). This is the first work to use classic backtracking techniques in a distributed setting. These algorithms can solve a broad class of distributed problems including resource allocation, scheduling, and truth maintenance. Distributed backtracking techniques are the basis for much of the work on distributed problem solving. This is due to the expressiveness of the CSP framework.

The work of [121] is extended to distributed optimization in [83]. The Distributed Optimization Problem (DCOP) is the problem of minimizing a quasi-linear objective function (i.e., a sum of nonlinear terms) given that the variables of discourse are distributed among a group of communicating agents. The DCOP can be solved using a cousin of Distributed Backtracking called the Asynchronous Distributed Optimization (ADOPT) Algorithm. This work [83] is the first to address distributed optimization with this level of generality.

Resource Constraints are included in the the DCOP formulations of [8, 78]. The

work of [8] developed a version of the ADOPT algorithm that is able to prune regions of the solution space that violate hard constraints. In [78], the Resource Constrained DCOP is introduced that considers that certain variables represent the use of particular limited resources. The objective of the optimization is to minimize the same quasi-linear objective function of the original DCOP subject to the satisfaction of the resource constraints. The complexity of the algorithm in [78] is subsequently improved in [79].

Privacy of agents' information is a concern in modern distributed problem solving research [35, 122]. Agent privacy can be of concern in situations where relinquishing information can decrease an agent's competitive position. In scenarios of negotiation where parties have complex needs and desires the constraints may contain information on these needs and desires that the parties may not want to reveal. The works of [35] and [122] provide modifications to existing distributed constraint satisfaction and distributed optimization algorithms that help to reduce the amount of private information shared during the solution process.

The method of Asynchronous Partial Overlay (APO) is a mediation-based algorithm for solving DCSPs [76]. The method of APO is a new paradigm for distributed constraint satisfaction. In contrast to traditional distributed backtracking techniques, agents maintain two lists, *good_list* and *agent_view*, that contain information about the variables, variable domains, and constraints that involve the agents' own variables. Conflicts are resolved using mediation. An agent requests to be mediator when it recognizes a conflict in the current variable values. Mediators are selected based on the size of their *good_list*, i.e., how much knowledge they have. Mediators are responsible for resolving conflicts. The algorithm is correct and complete.

The work of [99] presented modern DCSP solution techniques and discussed several open problems in the DCSP community including, privacy of agent data, exploiting degrees of problem distribution to reduce message and time complexity, failure in DC-

SPs, and uncertainty in the knowledge of constraints. This work also described the high-level developments of modern train scheduling problems using the CSP framework.

The Distributed Backtracking-based methods of [121, 120, 83] assume that agents sharing involvement in constraints are neighbors. As such, these agents can communicate directly. The assumptions on the nature of the communication are equivalent to an acknowledgement-based communication link with delay (i.e., messages are received in the order they are sent, are subject to arbitrary finite delay, and are not lost or corrupted). These assumptions are realistic in many scenarios and have led to expressive distributed algorithms for solving the DCSP and DCOP. These assumptions incur a penalty in the communication links required; this penalty is related to the expressiveness of the constraints used to describe the problem. This dependence is discussed in detail in Section 4.1.7.

The current work extends the DCSP formalism for use in constrained distributed scheduling. The DCSP can be stated as follows. Consider a set of N processes, $Processes$, each in control of a variable $v_i \in \mathcal{D}_i$, $i = 1, \ldots, N$. The processes communicate using acknowledgement-based communication links. The communication topology is specified by the process graph (1.15). The neighborhood of a process $[p_i]$ on $\mathcal{G}_p$ is defined as

$$\mathcal{N}_{[p_i]} = \{[p_j] : \{[p_i], [p_j]\} \in \mathcal{E}_p\}. \tag{2.19}$$

Consider N_c constraints,

$$p_m : \mathcal{D}_1 \times \ldots \times \mathcal{D}_N \rightarrow \{false, true\}, m = 1, \ldots, N_c, \tag{2.20}$$

where the inputs to the constraint functions are the values of the variables and the constraints output $true$ if the values of the variables satisfy the constraint and output $false$ if they do not. The N processes trade messages to inform their neighbors of

the values of their variables. For the purposes of backtracking, a priority function

$$PR : Processes \rightarrow \mathbb{N}, \tag{2.21}$$

is defined. The priority function allows for processes to be ordered. This ordering is used in the distributed backtracking search to properly perform the operations of expansion and backtracking. The specification and detailed role of this function is discussed further in Section 4.2.

The following assumptions illustrate the difficulty of solving a CSP due to distributing the problem data over several processes. A process $[p_i]$ is assumed to know the following:

1. Its neighborhood on the communication graph $\mathcal{N}_{[p_i]}$,
2. Its priority $PR([p_i])$,
3. All constraints p_m that v_i is involved in,
4. And the domain of v_i, $\mathcal{D}_i$.

The processes send and receive messages $M \in \mathcal{M}$. The assumptions on message communication are as follows.

1. Messages may experience arbitrary, but finite delay.
2. Messages are not lost.
3. Messages may be duplicated.
4. Messages are not corrupted.

The DCSP is: given the distributed knowledge and authority of the processes, find values of the N variables such that all constraints evaluate $true$.

2.3.3 Distributed Computing

Distributed computing is the field that is concerned with the inter-operation of computers that operate asynchronously, and are subject to communication delays, process failure, communication failure, and distribution of control, [36]. A review of distributed computing is given in [36]. This review discusses the theoretical developments of distributed computing in comparison to the solutions needed and developed by practitioners. The work illustrates a disconnect between theoretical developments and the practical utility of these theories. The use of pessimistic models of process asynchrony, process failure, and communication failure can result in theoretical results that are illuminating and elegant, but that may have limited utility in practice. These results, [67, 37], often motivate the development of tools that are heuristic, but function effectively in practice [16, 15].

The fundamental result of [37] considered the commit problem (see Section 2.3.4). They show that for a completely asynchronous distributed system, agreement between processes is impossible in the presence of even one faulty process. In practice complete asynchrony is extremely pessimistic. In practice, algorithms are available for solving the commit problem under slightly more restrictive assumptions [68, 69, 15]. While algorithms to overcome faults are available, it can be more useful to have a more general formalism that guarantees properties on which other distributed algorithms can rely, [16]. In [16], it is shown that (distributed) failure detectors can be designed that can eventually determine which processes have failed and which have not. This work allows for the use of failure detectors, which satisfy certain properties, to be used to assist other distributed algorithms.

The following distributed problems and distributed algorithms to solve them are presented in [75]: electing a leader amongst several processes, constructing a breadth-first search tree, constructing a minimum spanning tree, constructing a maximal independent set, and resource allocation. Several of these problems are addressed

for synchronous and asynchronous settings. The following subsections review the problems of distributed commitment and distributed consensus.

2.3.4 Distributed Consensus

Agreement is a fundamental problem in distributed systems. Agreement problems primarily come in two types: 1) several processes are to agree on the value for a single discrete variable among several possible proposed values, and 2) several processes are to asymptotically agree on the value of a variable. We refer to the first case as commitment and the second as consensus. Algorithms that solve these two types of consensus problems operate very differently. Discussion of these problems and algorithms for their solution follow.

2.3.4.1 The Commit Problem

The problem of having multiple agents achieve consensus by commitment to the discrete value of a quantity is called the commit problem. Algorithms used to solve this problem are designed to ameliorate the effects of agent failure. In [66], the algorithmic and implementation challenges of such systems were detailed. The work of [37] proved the fundamental need for the agents of a distributed system to have access to at least a rudimentary clock in order to solve the commit problem. Here, we do not consider failure, but address the challenges of task assignment and task scheduling when the communication topology is arbitrary and only locally known.

Commitment is treated in [69, 15, 13, 75, 66] and [37]. The commit problem in distributed systems is motivated by the need for fault-tolerant algorithms for distributed databases. The commit problem is often stated as the coordinated attack problem [75]. In this problem, several generals must coordinate an attack. The generals must attack if possible and when they attack, they must all attack at once. The generals coordinate the attack by sending messages to each other via message couriers that

are unreliable in the sense that they may by captured or otherwise not deliver their messages. One of the most successful algorithms used to solve the commit problem is the Paxos Algorithm [69]. The algorithm is elegant and correct, but comes with its own implementation challenges which are discussed in [15]. The Paxos Algorithm is described briefly as follows.

Consider N processes with unique IDs where the processes are connected by unreliable communication links. All processes operate under the assumptions of the following partially synchronous model.

1. Messages may be lost, delayed for an arbitrary length of time, or duplicated, but are never corrupted.
2. Processes may fail by simply stopping and may recover.
3. Processes have incorruptible memory.
4. Processes know when they have failed and recovered.

The communication graph is complete and known by all processes. Values v with an associated proposal number, $m \in \mathbb{N}$ can be proposed, without loss of generality $v \in \mathbb{N}$. The following four conditions must hold to guarantee the safety (or correctness) and completeness of a commit algorithm.

1. Only a value that has been proposed by some process may be chosen.
2. Only one value may be chosen.
3. A process never learns that a value has been chosen unless it actually has been.
4. Eventually a proposed value is learned.

The Paxos algorithm uses three classes of processes: proposers, acceptors, and learners. In the context of distributed databases, proposers act on behalf of a client who

wants to modify a database. The role of acceptors is to approve the modification given that several proposers may be vying for different modifications. Learners are the memory of this distributed system; their role is to remember the values that are accepted by acceptors.

2.3.4.2 The Consensus Problem

In [92], the problem of consensus regarding time varying quantities is studied. The variety of consensus problems in the literature include consensus in the presence of time delays [96] and applications to distributed estimation [14]. A detailed survey of consensus problems and discussion of properties therein is given by [93]. Algorithms for distributed consensus work well when multiple agents must come to approximate agreement, e.g., in applications of estimation where approximate agreement is sufficient. Consensus algorithms seek results of an asymptotic nature. Typical consensus problems are for a set of agents to agree on the common values of a set of quantities. Often these quantities represent vehicle positions, velocities, or decision variables.

Communication models for consensus problems typically involve a set of agents as in (3.2), that are the nodes of a communication graph. The edges of the graph represent communication links between agents and can be assumed to be directed or undirected. In the case of a directed graph, the communication between agents is unidirectional and the communication graph is assumed to be strongly connected. That is, there is a directed path between any two vertices of the graph. For an undirected communication graph, communication is bidirectional and the communication graph is assumed to be connected. That is, there is an undirected path between any two vertices of the graph.

The basic consensus problem consists of the set of agents and a quantity x to be estimated by each of the agents. The consensus problem is for all agents to asymptotically output the same estimated value for x. Let $\mathcal{N}_{a_i}$ be the neighborhood

of agent $a_i \in \mathcal{A}$ on the undirected communication graph. Let x_i be the value of x approximated by agent $a_i \in \mathcal{A}$. An agent updates its estimate of x by an update law of the following form,

$$\dot{x}_i = -\sum_{j:a_j \in \mathcal{N}_i} \alpha_{ij}(t)(x_i(t) - x_j(t)), x_i(t_0) = x_{0_i}, \tag{2.22}$$

where for a static topology $\alpha_{ij}(t) = \alpha_{ij}$ and α_{ij} is a function of the time-invariant adjacency matrix of the communication graph. For agents communicating over an undirected, connected communication graph, an update law of this form results in so-called average-consensus. That is, for the asymptotic estimates of the agents,

$$\lim_{t\to\infty} x_1 = \ldots = \lim_{t\to\infty} x_{N_a} = \frac{\sum_{i:a_i \in \mathcal{A}} x_{0_i}}{N_a}. \tag{2.23}$$

The problem formulations and algorithms presented in this section address a wide range of problems for vehicle routing, task assignment and task scheduling, multi-agent, and distributed systems. This dissertation contributes to the centralized and distributed task assignment and task scheduling literature. This contribution involves (to varying degrees) the intersection of all of these subject areas. The contributions of this dissertation are further discussed in the context of these bodies of literature in the following section.

2.4 Limitations of Existing Literature

This section reviews the limitations of the existing literature that are of interest in this dissertation.

2.4.1 Centralized Task Assignment and Task Scheduling

Existing centralized formulations address multiple agent task assignment and task scheduling with precedence constraints, and minimum-time scheduling. However, existing approaches either incur a high computational burden, or only address a subset of these features. This limits the use of these methods on problem instances of practical size. The Tabu/2-opt heuristic developed here includes precedence constraints and mission time minimization at a reduced computational burden. This is accomplished by using a novel separation of the VRP solutions as task assignments and task schedules. The task assignments are optimized using a Tabu Search heuristic. The schedules are optimized using a recursive 2-opt heuristic that optimizes the total mission time. As a secondary objective the 2-opt heuristic is used to locally optimize individual agent routes. The combination of heuristics is shown to solve the version of the VRP stated here and is able to find feasible solutions to the optimization problem quickly.

2.4.2 Distributed Task Assignment and Task Scheduling

The existing unmanned air vehicle task assignment and task scheduling literature incorporates expressive mission constraints as in [33, 58, 34, 103]. These formulations compute the task assignments and task schedules in a centralized way for distributed execution. They solve for a solution centrally and then distribute the task assignment and task schedule while assuming the central computer can communicate reliably with all agents. They are inherently incapable of exploiting distributed computation because these formulations do not consider distributed-ness in the problem formulation.

The development of distributed task assignment and task scheduling algorithms that incorporate temporal and logical mission constraints has not yet been addressed. Also, no formulations address minimization of mission time subject to mission constraints while using an arbitrary communication network topology. The work of this

dissertation extends the distributed task assignment and task scheduling literature into these areas and provides tools for cooperative distributed task assignment and task scheduling with constrained communication. Section 2.5 presents more detailed comparisons to the methods that most closely address the features of interest in this dissertation.

2.5 Comparisons With Existing Centralized Methods

This section compares several centralized methods of formulating task assignment and task scheduling problems. These frameworks and algorithms address important features of the problems posed and solved in this dissertation.

Single TSP Formulations can be solved with great efficiency using tools that can be leveraged in solving the mTSP and VRP. Mixed Integer Linear Programming Formulations have been used to design task assignments and task schedules for the mTSP. Tree Search problem formulations can express a wide variety of VRPs and Branch and Bound Algorithms have been used on several occasions to solve VRPs. Precedence Constrained TSPs (PCTSP) incorporate precedence constraints into the single TSP can be expressed as linear programs and solved using Branch and Cut Algorithms. Temporal and logical constraints can be incorporated into VRP formulations. This section compares the above methods with the VRP formulation used here and Tabu/2-opt Heuristic used to solve it.

2.5.1 TSP, PCTSP, and mTSP Comparison

The TSP, mTSP, and PCTSP are formulations that (individually) incorporate several features of the VRP the current work addresses. The classic formulation of the TSP is for $N_a = 1$ and $N_t > 0$. The formulation of the mTSP considers $N_a > 0$ agents and $N_t > 0$ tasks, or cities to visit. The classic TSP formulation cannot be used directly to address problems of multiple vehicle task assignment and task

Problem	Algorithm/ Method	Assignment	Scheduling	Temporal Constraints	Logical Constraints	Large instances	Near optimality	Optimize Mission Time
TSP	Lin-Kernighan	No	Yes	No	No	Yes	Yes	Yes (single agent)
Precedence-TSP	B&Cut	No	Yes	Yes	No	High complexity	Yes	No
mTSP	Integer Linear Program	Yes	Yes	Only for same agent	No	Possibly high complexity	Yes	No
Temporal/ Logical Missions	MILP	Yes	Yes	Yes	Yes	High complexity	Yes	Yes
Any VRP	B&B	Yes	Yes	Yes	Yes	High complexity	Yes	Yes
Precedence-constrained minimum time VRP	Tabu/2-opt	Yes	Yes	Yes	No	Yes	Within 25% of optimal (Average)	Yes

Figure 2.1: Comparison of centralized algorithms for task assignment and task scheduling.

scheduling. However, with modification, tools for solving the classic TSP can be used for solving mTSP instances [42]. Several mTSP formulations are available in [55] that can leverage the efficient tools of linear programming. However, these formulations are unable to optimize for mission time and are not capable of including precedence constraints when such constraints relate tasks performed by separate agents. The formulation of this work can express, precedence constraints between tasks that are performed by different agents, and a mission time objective function. The Tabu/2-opt heuristic designed here is able to solve this problem while searching through only feasible solutions (i.e., only solutions that obey precedence constraints).

The PCTSP formulation of [3] incorporates precedence constraints into a classic TSP problem and is solved using a Branch and Cut method. The algorithm iteratively solves a linear program. Branch and Cut relies on generating cutting plane constraints from precedence constraints. These constraints are generated when the algorithm discovers a portion of the solution space that violates the precedence constraints. The method then proceeds augmented with the additional constraints. This technique is able to leverage traditionally efficient linear programming tools, but it relies on a linear

expression for the objective function and is only a single TSP formulation. The work of [102] is able to ameliorate this worst-case exponential increase in the number of constraints and provide polynomial bounds on the number of cutting plan constraints generated. This formulation only addresses the precedence constrained single TSP. It is not able to express VRPs with other types of constraints and objective functions.

2.5.2 Branch and Bound and MILP-Based Methods Comparison

Tree Search formulations can be used to solve a variety of problems in vehicle routing [70]. Any problem that can be expressed as a search-tree can be solved using Branch and Bound B&B algorithms. Because the core technique used is to enumerate possible solutions, Tree Search methods can incorporate precedence, logical, capacity, and other types of constraints. This same expressiveness can be used to incorporate objective functions that represent mission time, risk, etc. The strength of B&B methods is that they can be very efficient if tight bounds are available for the particular application at hand. The weakness of B&B methods is that they can be as inefficient as exhaustive search if bounding information is not available to help prune the search space. For this reason B&B is a less viable option for larger scale VRPs. Contrary to these limitations, the Tabu/2-opt heuristic used here is able to directly generate and optimize over a space of feasible solutions. This allows the heuristic to provide feasible solutions quickly.

Mixed Integer Linear Programming formulations are able to express problems that have an objective function and constraints that are linear in the problem variables. These problem formulations are able to incorporate discrete and continuous variables. In VRP formulations the discrete MILP variables typically represent decision variables indicating which vehicles will perform each of the tasks. The continuous variables represent time. The MILP formulations suffer from the same scalability issues that reduce the practical effectiveness of Branch and Bound methods. The size of MILP

problems can scale exponentially with the number of tasks and agents.

2.6 Comparisons With Existing Distributed Methods

This section gives a comparison with several distributed methods this dissertation considers to be representative of important and related classes of task assignment and task scheduling frameworks and algorithms. These frameworks and algorithms address features we believe are important to the problems detailed and solved in this dissertation.

The assignment problem presented in [7] is solved using the Classic Auction Algorithm. The nonlinear assignment problem of [19] incorporates a large class of objective function and is solved using the Consensus-Based Bundle Algorithm and the Consensus-Based Auction Algorithm. The Auction-Based Distributed Scheduling Algorithm of [27] solves a precedence constrained task assignment and task scheduling problem. Constraint Satisfaction Problem Frameworks are an expressive way to incorporate constraints into task assignment and task scheduling problems. These types of problems can be addressed using Distributed Backtracking methods. Lastly, the Distributed Constraint Optimization Problem Framework can address a broad class of distributed optimization problems. The table of Figure 2.2 presents comparison with these formulations.

2.6.1 Task Assignment Formulations Using Auction-Based Methods

The assignment problem of [7] is to find a task assignment TA that is feasible with respect to capability constraints, i.e., $(t, TA(t)) \in \mathit{Capability}$, $t \in \mathcal{T}$. The Classic Auction Algorithm solves a maximization problem where $N_a = N_t$ with the following objective function,

$$J(TA) = \sum_{(i,j) \in TA} b_{ij}, \tag{2.24}$$

Problem	Algorithm/ Method	Assignment	Scheduling	Communication Constraints	Temporal Constraints	Logical Constraints	Kinematics	Optimize Mission Time	Constrained Mission Time Opt.
Classic assignment problem	Auction (classic)	Yes	No	No	No	No	Yes	No	No
Bundle assignment	Auction (CBBA)	Yes	Yes	No	No	No	Yes	No	No
Distributed machine scheduling	ABD Scheduling	Yes	Yes	No	Yes	No	No	No	No
DCSP	DBT	Yes	Yes	Increased complexity and communication links	Yes	Yes	Increased complexity	No	No
DCOP	Adopt	Yes	Yes	Increased complexity and communication links	Yes	Yes	Increased complexity	Yes	No
CDAP/MADSP	SB/DNSB	Yes	Yes	Yes	Yes	Yes	Increased complexity	Yes	Yes

Figure 2.2: Comparison of distributed algorithms for task assignment and task scheduling.

where b_{ij} is the benefit of assigning task $t_i \in \mathcal{T}$ to agent $a_j \in \mathcal{A}$. The optimization problem can be stated as

$$\max_{TA \in \mathcal{A}^{\mathcal{T}}} J(TA) \tag{2.25}$$

$$\text{s.t. } TA \subseteq Capability$$

The objective function here is linear in the assignments. While extremely versatile, the Classic Assignment Algorithm cannot be used to find a task assignment that satisfies the constraints of (1.9).

The work of [19] addresses an assignment problem similar to that of (2.25), where $N_t \geq N_a$ and the b_{ij}'s are not constants, but are functions of the set of tasks that are assigned to the agent $a_j \in \mathcal{A}$. The Consensus Based Bundle Algorithm (CBBA) developed in [19] is an auction algorithm built on several extensions to the Classic Auction Algorithm.

The CBBA is able to find assignments that minimize an objective function that need only be a non-negative function of the assignments and for which the b_{ij}'s must satisfy the property of having diminishing marginal gain (DMG). This means that

the value of a task should not increase as other elements are added to the set before it, [19]. This is not the case for the objective function representing the constraints of (1.9). The violation of this property is due to the fact that assigning additional tasks to a single agent may satisfy additional clustering constraints not yet satisfied, thus improving the value of the assignments already made with respect to this agent. In addition to our problem's violation of the DMG assumption, we require that all clustering constraints be satisfied. The CBBA and CBAA of [19] have guarantees of within 50% of optimal. In the current work, the CDAP is solved by converting the communication constraints into an objective function to be minimized. If the work of [19] were used to solve this problem we could not guarantee the global minimization of this objective function that we need to satisfy all clustering constraints.

The work of [27] is an elegant approach to solve a distributed task assignment and task scheduling problem. This work considers precedence constrained task scheduling to minimize an objective function that is linear with respect to the scheduled times of the tasks. The coefficients are unlike those in (2.24) in that they depend on the time when the task is performed. The method detailed in [27] is used by agents operating on a complete communication network. An auction is set up to minimize the linear objective function subject to precedence constraints. The auction uses a novel dual auction method where the constraints are relaxed in a Lagrangian sense. The tasks bid for time slots and agents to perform them. The agents bid to minimize the used of their own capacity. The price that the tasks must bid for the time slots is related to this capacity. This method finds an optimal solution for the problem addressed, but leverages the linearity of the objective function and linearity of the constraints. We do not require this linearity in the current work.

2.6.2 Distributed Constraint Satisfaction and Distributed Constraint Optimization Comparison

With regards to including clustering constraints in a Distributed Constraint Satisfaction Problem formulation, to include the constraint that those agents assigned tasks that share involvement in a constraint,

$$\sum_{m=1}^{N_c} \mid Capability(\mathcal{T}_m) \mid \tag{2.26}$$

additional binary constraints must be included in the formulation.

There is the additional requirement that a task assignment be a mapping. The Stochastic Bidding Algorithm inherently ensures this requirement is met. Incorporating this into a DCOP framework would require N_t constraints whose order is given by $N = |Capability(t_i)|$, $i = 1, \ldots, N_t$.

If distributed backtracking is used to find a schedule that obeys clustering and mapping constraints, the worst-case complexity would be $\mathcal{O}(N^{N_t})$. Backtracking is efficient when the constraints are of low order (e.g., order 1 or 2); as the order of the constraints increases, Backtracking becomes less efficient. The clustering constraints can be decomposed into binary constraints, but the mapping constraint for a task $t \in \mathcal{T}$ has order $|Capability(t)|$.

It is possible to formulate the CDAP and the MADSP using Distributed Constraint Optimization. The details and consequences of such a formulation are given below. The distributed constraint optimization problem (DCOP) presented in [83] generalizes the notion of constraint satisfaction. Typically, a constraint satisfaction problem is to find the arguments for constraints that produce all *true* values. The work of [83] generalizes the constraints to be integer functions of integer variables and seeks to find the arguments that minimize the sum of the outputs of these functions. This DCOP formulation considers $N > 0$ variables v_i and their values $d_i \in \mathcal{D}_i$,

$i = 1, \ldots, N$. The objective function for the DCOP formulation is

$$J(d_1, \ldots, d_N) = \sum_{m=1}^{N_c} f_m(d_1, \ldots, d_N) \tag{2.27}$$

where $f_m : \mathcal{D}_1 \times \ldots \times \mathcal{D}_N \rightarrow \mathbb{N}$, $m = 1, \ldots, N_c$. The DCOP formulation includes optimization constraints as a subset of the functions f_m. The DCOP formulation in [83] considers the functions f_m to have only two arguments; however, it is conjectured that this can be generalized as above. The following analysis conservatively assumes that this generalization holds.

For this analysis, we treat our Minimum-Time Arbitrarily-Constrained Distributed Scheduling Problem (MADSP) with the DCOP framework. The objective function representing the maximum finish time for the schedule can be represented as the function $f_0 : \mathbb{T}^{\mathcal{T}}_{max} \rightarrow \mathbb{N}$, where

$$f_0 = \max_{t \in \mathcal{T}} TS(t) \tag{2.28}$$

and the optimization constraints are

$$f_m : \mathbb{T}^{\mathcal{T}}_s \rightarrow \{0, 1\} \subset \mathbb{N}. \tag{2.29}$$

For these constraints, let $f_m = 0$ when the constraint is *satisfied* and $f_m = 1$ when the constraint is *violated*. The reason for the difference in the definitions of satisfied and violated will be clear shortly. In the DCOP framework, the constraint violations and the objective function are minimized simultaneously as

$$\min_{TS \in \mathbb{T}^{\mathcal{T}}_s} f_0(TS) + \sum_{m=1}^{N_c} f_m(TS). \tag{2.30}$$

By definition, a schedule TS is feasible if $f_m(TS) = 0$, $m = 1, \ldots, N_c$. Addition-

ally, a schedule TS is feasible if and only if

$$\sum_{m=1}^{N_c} f_m(TS) = 0. \tag{2.31}$$

A schedule TS^* is optimal, with respect to minimizing f_0, if $\forall TS \in \mathbb{T}^{\mathcal{T}}_{max}$, $f_0(TS^*) \leq f_0(TS)$. The OptDNSB Algorithm finds a feasible schedule if one exists; this schedule is provably optimal. The following development shows that, for a class of constrained optimization problems, the Adopt Algorithm is not guaranteed to return feasible optimal schedules.

Consider a schedule TS', that is not feasible, that is optimal with respect to $f_0(TS) + \sum_{m=1}^{N_c} f_m(TS)$. That is,

$$\forall TS \in \mathbb{T}^{\mathcal{T}}_{max}, f_0(TS') + \sum_{m=1}^{N_c} f_m(TS') \leq f_0(TS) + \sum_{m=1}^{N_c} f_m(TS). \tag{2.32}$$

Also, let $f_0(TS') < f_0(TS^*) - 1$, $f_1(TS') = 1$, and $\sum_{m=2}^{N_c} f_m(TS') = 0$. That is, TS' is a solution to the DCOP, and one that the Adopt Algorithm can return as a solution. However, TS' is not feasible. This behavior is intuitively reasonable; for example, in the context of scheduling improvements in the total schedule time can be gained by violating precedence constraints.

2.6.3 Communication Consequences

It is common in the distributed systems and distributed problem solving literature to assume that agents are able to, or only required to, communicate with a subset of all available agents, [83, 111, 120, 121]. This assumption is useful because it does not require that a complete network be maintained during the assignment and scheduling process. In the absence of this assumption, the communication load can be increased. The function in Equation (2.28) involves all of the tasks in $\mathcal{T}$. By the

assumption that agents can communicate directly if their tasks share involvement in a function, f_i, $i = 0, \ldots, N_c$, agents assigned any task must be able to communicate by acknowledgement-based link.

Several commonly cited distributed constraint satisfaction techniques carry the name *asynchronous* [120, 83]. These methods use the following important assumption; messages are received in the order in which they are sent. This assumption can often be guaranteed in practice; this is accomplished using acknowledgement-based communication protocols (e.g., TCP/IP, [109]). While useful, this assumption restricts the distributed algorithm to being partially synchronous rather than asynchronous. Strict definitions of synchronous and asynchronous distributed algorithms are given in [37, 75]; these are used here. As such, we do not refer to algorithms that follow the above assumptions as asynchronous.

The literature review of this chapter reviews the areas of Vehicle Routing, Task Assignment and Task Scheduling, and Multi-Agent and Distributed Systems. These areas of the literature provide context for the remainder of the dissertation. Chapter III develops and solves the problem of multiple vehicle routing where mission time is minimized subject to precedence and vehicle kinematic constraints. A centralized point of view is taken in this chapter. Chapter IV develops and solves two problems: the first is that of distributed communication-constrained task assignment, the second leverages these results to solve the problem of constrained minimum-time task scheduling. A multi-agent and distributed systems point of view is taken when formulating the distributed task assignment and task scheduling problem.

CHAPTER III

Centralized Task Assignment and Task Scheduling Problem Formulation

This chapter develops the centralized task assignment and task scheduling problem of this dissertation. The formulation of this chapter contributes a way to formulate a relevant class of mission optimization problems [104]. There are several important features to note; this problem includes: capability constraints on the task assignments, precedence constraints between tasks, multi-agent task cooperation (synchronized agent to task arrival times), vehicle kinematic constraints, and minimization of mission time. This problem is of particular interest because our research considers the planning of military missions [104], where these types of constraints occur naturally. The Tabu/2-opt Algorithm developed to solve this problem also has several features of note. The algorithm has polynomial time complexity. It can output feasible solutions in polynomial time and in fractions of a second. It minimizes mission time with the added optimization of individual agent routes. And it converges to solutions within (average) 25% of optimal in seconds for medium size problems. The Tabu/2-opt heuristic is a so-called greedy repair heuristic. That is, it iteratively improves an initial candidate solution to generate a solution that is closer to optimality than the initial solution. The discussion below formulates this centralized task assignment and task scheduling problem with application to multiple vehicle routing.

This chapter also introduces a technique that is able to quantify the quality of candidate solutions to combinatorial optimization problems for which there is no analytical solution quality guarantee. This technique takes a unique approach, stochastic characterization of the solution set of a problem, to gain insight into the cost distribution of candidate solutions. The technique then applies a transformation to the sampled distribution that permits the use of standard Gaussian tools to provide relative solution quality measures.

To formulate task assignment and task scheduling problems, we need several abstractions. The notions of tasks and agents are used to abstract the specific tasks to be completed and the machines that will perform the specific tasks. Formally, the set of tasks is

$$\mathcal{T} = \{t_1, \ldots, t_{N_t}\}, \tag{3.1}$$

where $N_t > 0$ is the number of tasks. Formally, the set of agents is

$$\mathcal{A} = \{a_1, \ldots, a_{N_a}\}, \tag{3.2}$$

where $N_a > 0$ is the number of agents.

The capabilities of the agents to perform the various tasks are described using a relation from $\mathcal{T}$ to $\mathcal{A}$, i.e., a subset of their Cartesian product:

$$Capability \subseteq \mathcal{T} \times \mathcal{A}. \tag{3.3}$$

A pair $(t, a) \in \mathcal{T} \times \mathcal{A}$ is in *Capability* if and only if task t *can be performed by* agent a. Without loss of generality, we assume that the range of *Capability* is $\mathcal{A}$. In other words, we assume that each agent is capable of performing at least one task. Informally a *task assignment* indicates which agent or agents will perform a task. Depending on the formulation, a task assignment may be a relation or a mapping.

This is a modeling decision. This dissertation uses two formulations of the problem: in the first a task assignment is a relation; in the second, a task assignment is a mapping. A task assignment that is defined as a relation is formally given as

$$TA \subseteq \mathcal{T} \times \mathcal{A} \tag{3.4}$$

and an a task assignment that is defined as a mapping is formally given as

$$TA : \mathcal{T} \rightarrow \mathcal{A}. \tag{3.5}$$

This distinction lies in the uniqueness of the assignment. For a relational task assignment, a task is assigned to a set of agents. For the case of a mapping, we require that each task be assigned to one and only one agent. This mapping need not be injective, i.e., an agent may be assigned several tasks. This mapping need not be surjective, i.e., an agent may not be assigned any task at all. Note that there is no loss of generality in requiring that a task assignment be a mapping: if a task requires several agents, it should be split into subtasks requiring one agent each. The choice of how to model the task assignments is dictated by the algorithm used to solve the task assignment problem.

A task assignment is called *feasible with respect to capability* if $TA \subseteq Capability$, i.e.,

$$(t_i, TA(t_i)) \in Capability, i = 1, \ldots, N_t. \tag{3.6}$$

In practice, this means that each task is assigned to an agent that is capable of performing it.

Informally, a schedule indicates whether a task will be performed and when it will be performed. A task *schedule* is a mapping from the set of tasks $\mathcal{T}$ to a set of non-negative *finish times* $\mathbb{T}_s$. We assume that all tasks are assigned, but that tasks may be

scheduled such that they are not executed. This allows for agents to choose whether or not to perform tasks if permitted (or required) by scheduling constraints. This can be modeled by specifying that a finish time in $\mathbb{T}_s$ that (without loss of generality) equals zero represents a task not executed. The finish time of zero is chosen because a task would have to be reached in zero travel time and have zero duration for it to finish execution at time zero. This is unlikely in the current framework, hence the zero finish time is reserved. The set of finish times may be finite or not. Formally, a schedule is a mapping

$$TS : \mathcal{T} \rightarrow \mathbb{T}_s. \tag{3.7}$$

Additionally, TS need not be injective nor surjective.

3.1 Constraints

Constraints are used to restrict the set of allowable task assignments and task schedules. In general, constraints involve task assignments and task schedules. Informally, a constraint is a function that tells whether or not a task assignment and task schedule are acceptable. Physically, constraints may represent precedence restrictions, capability restrictions, capacity limits, timing restrictions, communication restrictions, etc. Formally, a constraint is a function

$$p_{AS_m} : \mathcal{A}^{\mathcal{T}} \times \mathbb{T}_s^{\mathcal{T}} \rightarrow \{false, true\}, m = 1, \ldots, N_{cl} \tag{3.8}$$

where N_{cl} is the number of constraints. Given a task assignment TA and a task schedule TS, a constraint p_m is *violated* if $p_{AS_m}(TA, TS) = false$ and p_m is *satisfied* if $p_{AS_m}(TA, TS) = true$.

Assignment constraints restrict which agents tasks can be assigned to. That is, assignment constraints are only a function of the task assignments TA. Formally,

assignment constraints are defined as

$$p_m : \mathcal{A}^{\mathcal{T}} \rightarrow \{false, true\}, m = 1, \ldots, N_{as}, \tag{3.9}$$

where $N_{as} > 0$ is the number of assignment constraints. The assignment constraints in (3.9) are general. That is, any assignment constraint that can be represented as a predicate that evaluates either $false$ or $true$ is captured by (3.9).

Informally, scheduling *constraints* are used to characterize the allowable schedules. Scheduling constraints are formally defined as functions that map from the set of schedules to the set $\{false, true\}$. The number of constraints is N_{cl}. These constraints are defined as

$$p_m : \mathbb{T}_s^{\mathcal{T}} \rightarrow \{false, true\}, m = 1, \ldots, N_s, \tag{3.10}$$

where $N_s > 0$ is the number of scheduling constraints. The scheduling constraints in (3.10) are general in the same sense as (3.9).

3.2 Constraint Satisfaction Problems and Backtracking

The Constraint Satisfaction Problem (CSP) is the problem of finding values of several variables that satisfy predicate constraints that are a function of the variables. The CSP formalism has been used to solve scheduling problems in [98, 97]. Formally, a CSP is given by several variables with their respective universes of discourse and predicate constraints, defined on these universes of discourse, to be satisfied. The variables of the problem are

$$v_i \in \mathcal{D}_i, i = 1, \ldots, N_v \tag{3.11}$$

where $N_v > 0$ is the number of variables and $\mathcal{D}_i$ is a finite, typically discrete, universe of discourse for the variable v_i.

The constraints are

$$p_m : \mathcal{D}_1 \times \ldots \times \mathcal{D}_{N_v} \rightarrow \{false, true\} \tag{3.12}$$

CSPs can be solved efficiently using variants of Backtracking. The traditional backtracking algorithm is detailed as follows.

Given an ordered set $\mathcal{D}$, define a set of sequences $\mathcal{D}^{N_v}$ of length less than or equal to N_v. Let $vs_i \in \mathcal{D}^{N_v}$ be one such sequence of length i, with elements $vs_i(j) \in vs_i$, $j = 1, \ldots, i$. Similar to (3.12), define constraints $p_m : \mathcal{D}^{N_v} \rightarrow \{false, true\}$ on sequences $vs_i \in \mathcal{D}^{N_v}$, $m = 1, \ldots, N_c$. The problem here is to find a sequence $vs_i \in \mathcal{D}^{N_v}$ such that $i = N_v$ and $p_m(vs_i) = true$, $m = 1, \ldots, N_c$. This can be done using a backtracking algorithm [95].

Define the set $untried_i \subseteq \mathcal{D}$. Define the function $expand : \mathcal{D}^{N_v} \times \mathcal{D} \rightarrow \mathcal{D}^{N_v}$ that accepts as input, a sequence vs_i and $untried_i$ and returns a new sequence $vs_{i+1} \in \mathcal{D}^{N_v}$.

Define the function $backtrack : \mathcal{D}^{N_v} \rightarrow \mathcal{D}^{N_v}$ that accepts as input, a sequence vs_{i+1} and returns the sequence vs_i.

Data: $vs_0 = \emptyset$
1 $untried_1 = \mathcal{D}$
2 **while** $i < N_v$ *or* $\exists m : p_m(vs_i) = false$ **do**
3 **if** $untried_1 = \emptyset$ **then**
4 no solution exists
5 *return* $vs_i = \emptyset$
6 **else if** $\exists m : p_m(vs_i) = false$ **then**
7 $vs_{i-1} := backtrack(vs_i)$
8 **else**
9 $vs_{i+1} := expand(vs_i, untried_i)$
10 $untried_i = untried_i \setminus vs_{i+1}(i+1)$
11 $untried_{i+1} = \mathcal{D}$
12 **end**
13 **end**
Result: vs_i

Algorithm 1: Classic Backtracking Algorithm: performs Backtracking given a set variables, their domains, and the constraint functions to be satisfied by the variable values. The algorithm either outputs a solution, or that no solution exists.

The procedure in Algorithm 1 performs backtracking. If all values in $\mathcal{D}$ are exhausted for vs_1, there is no sequence $vs \in \mathcal{D}^{N_v}$ that can satisfy the given constraints. The expansion of new, possibly feasible, solutions is performed at line 9 in Algorithm 1. The removal, or pruning of subsequent portions of the set $\mathcal{D}^{N_v}$ that do not contain a feasible solution is done during backtracking at line 7 of Algorithm 1.

3.3 Nonlinear Versus Linear Objective Functions

Constraint satisfaction is essential; but optimization of an objective function is often important. That is, we often desire to formulate task assignments and task schedules that will (upon execution) optimize a resource of interest. This is often formulated as minimizing a chosen objective function. The objective function can represent: distance (e.g. distance traveled by one or more vehicles); time (e.g. time taken to complete several tasks [50]); or even risk (e.g. risk associated with executing a military mission [34]).

One of the simplest and most useful nonlinear objective functions is *mission time*. Consider the set of tasks in (3.1). The set of finish times for these tasks are given by the schedule as $\{TS(t_1), \ldots, TS(t_{N_t})\}$. One form of mission time is

$$J(TS) = \max_{t_i \in \mathcal{T}} TS(t_i). \tag{3.13}$$

Minimizing the objective function (3.13), i.e., solving

$$\min_{TS \in \mathbb{T}_s^{\mathcal{T}}} J(TS) \tag{3.14}$$

is important because this objective represents the time needed to complete all tasks. The objective function (3.13) is a nonlinear function of the variable of discourse, i.e., the schedule TS. Nonlinear objective functions present difficulty because, while

linearity can often be exploited to search the solution space efficiently, nonlinearity often defeats traditional linear tools. For problems with general nonlinear objective functions (i.e., problems where no properties of the objective function are assumed *a priori*) variants of Backtracking can be used to guarantee an optimal solution.

One of the simplest and most useful linear objective functions is a sum of benefits resulting from a task assignment TA. The set of agents assigned to each of the tasks $t_i \in \mathcal{T}$ is $\{TA(t_1), \ldots, TA(t_{N_t})\}$. Let $b_{ij} \in \mathbb{R}$ be the benefit of assigning task t_i to agent a_j. The objective function

$$J(TA) = \sum_{(i,j):(t_i,a_j)\in TA} b_{ij} \tag{3.15}$$

is the total benefit resulting from the task assignment TA. Maximizing the objective function (3.15), i.e., solving

$$\max_{TA\in\mathcal{A}^{\mathcal{T}}} J(TA) \tag{3.16}$$

represents the maximization of the total benefit resulting from the task assignment. The objective function in (3.15) is a linear function of the variable of discourse, i.e., the task assignments TA. Linear objective functions are less expressive. The objective function J must be a linear combination of the discourse variables.

The task assignment and task scheduling problems presented in this dissertation are problems with nonlinear objective functions. There are several reasons for this. The VRP detailed in Chapter III is a constrained task assignment and task scheduling problem where the objective is to minimize mission time. This is the objective function of (3.13), and is nonlinear in the schedule. In practice, this accomplishes the mission in as little time as possible while obeying all mission constraints. The constraints of the assignment problem detailed in Chapter IV are nonlinear in the assignments. This nonlinearity is the result of clustering constraints that bind tasks and the agents that are assigned to them. The objective function of the scheduling

problem detailed in Chapter IV is the objective function of (3.13) and is nonlinear in the schedule.

A variant of the vehicle routing problem is developed here that includes capability constraints and precedence constraints between tasks. The objective in this problem is to minimize mission time. Unlike the VRP with pick-up and deliver constraints, here, the precedence constraints apply to tasks that may be performed by different agents. In this case the routes must be coordinated so that the timing constraints are obeyed. To achieve this coordination, agents may vary their velocity between zero and a maximum velocity. The capabilities of the agents are heterogeneous; the agents may only be able to perform a subset of tasks and may travel with different maximum velocities.

The Tabu/2-opt Heuristic is developed to solve this VRP. The algorithm finds feasible solutions in fractions of a second, finds low cost solutions in seconds for medium sized problems, and finds low cost solutions in minutes for large problems. The Tabu/2-opt Heuristic is a gradient-based search heuristic. The heuristic uses a two-stage optimization; it optimizes the routes of the agents, and then optimizes the task assignments. This approach is also able to minimize an important secondary objective: the 2-opt part of the heuristic gives minimum length individual agent routes. This minimization follows the minimization of mission time subject to precedence constraints.

3.4 Vehicle Routing Problem Formulation

Consider a VRP where N_a agents are to complete N_t tasks. Assignment constraints restrict the possible assignments of agents to tasks and completion order constraints restrict the possible order which tasks can be performed. The sets of agents and tasks are $\mathcal{A}$ and $\mathcal{T}$ respectively, as in (3.1) and (3.2).

The notions of precedent and dependent tasks are used here to denote the set of all

tasks that must occur earlier in time or later in time, respectively, than a given task. The formulation of this chapter considers tasks of zero duration, hence, consideration of a single occurrence time for each task is sufficient. The occurrence time of a task t_j is $TS(t_j)$. The precedent and dependent sets are,

$$\mathcal{P}_j = \{t_k \in \mathcal{T} \mid TS(t_k) < TS(t_j)\}, j = 1, \ldots, N_t, \tag{3.17}$$

$$\mathcal{D}_j = \{t_k \in \mathcal{T} \mid TS(t_k) > TS(t_j)\}, j = 1, \ldots, N_t, \tag{3.18}$$

and capture all transitivity relationships. For this version of the VRP, the set $\mathcal{P}_j$ is empty for at least one task and the set $\mathcal{D}_j$ is empty for at least one task. The set $\mathcal{P}_j$ is used with $\mathcal{D}_j$ to quickly determine the feasibility of a given problem instance, as addressed in (3.28). The sets $Capability(t_j)$ and $TA(t_j)$, $j = 1, \ldots, N_t$, are the set of agents capable of performing task t_j and the set of agents assigned to perform task t_j respectively. Let TA and $\mathcal{C}o$ specify the assignments of vehicles to tasks and the order in which tasks are completed. A solution to the task assignment problem is given in the form of a task assignment TA and a partially ordered set giving the order in which the tasks should be completed, $\mathcal{C}o$. We require that a task assignment be a relation, as opposed to a mapping, as described in Chapter I. A solution to the task assignment problem is given by TA and $\mathcal{C}o$, where

$$TA \subseteq \mathcal{T} \times \mathcal{A}, \tag{3.19}$$

$$\mathcal{C}o = (u_1, \ldots, u_{N_t}). \tag{3.20}$$

Here, u_i represents the task that is placed at position i in the partial ordering. The arrival time of an agent a_i at a task t_k is $TS(t_k)$. The arrival time of agent a_i at its final location is $T_{ei}(TA, \mathcal{C}o)$. The optimization problem is,

$$\min_{TA,\mathcal{C}o} \{\max_i T_{ei}(TA, \mathcal{C}o)\} \text{ s.t.} \tag{3.21}$$

$$TS(t_k) > TS(t_j), \quad \forall t_j \in \mathcal{P}_k, \quad k = 1, \ldots, N_t, \tag{3.22}$$

$$TS(t_k) > T_{ki}, \quad \forall a_i \in TA(t_k), \quad k = 1, \ldots, N_t, \tag{3.23}$$

$$TA(t_j) \subseteq Capability(t_j), \quad j = 1, \ldots, N_t. \tag{3.24}$$

The objective is to minimize the mission time (3.21), that is, the time for the last agent to complete its final task. This objective function is a function of the assignments of agents to tasks, the order of completion of the tasks, and the vehicle kinematics. Note that only the set $\mathcal{P}_k$ is used in the problem specification, $\mathcal{D}_k$ is used later in (3.28) to check whether the problem can be solved by the heuristic method developed in this chapter. The operational constraints restrict agents to only perform tasks after all precedence constraints have been met (3.22) and after every agent assigned to the task has arrived (3.23). These constraints also restrict the agents permitted to be assigned to a task (3.24). This problem is of particular interest because our research considers the planning of military missions [104], where these types of constraints occur naturally. It is also appropriate to minimize the total duration of the mission. This formulation is flexible enough to allow agents to cooperate to complete individual tasks if this helps minimize the cost.

Recall that the vehicles are assumed to obey steerable unicycle kinematics, operate in the Euclidean plane, which is a reasonable approximation if the tasks are far apart compared to the actual turn radius of the vehicles. The vehicle velocities are limited by $V_i \in [0, V_{\max_i}]$, where $V_{\max_i}$ is the maximum travel velocity of agent a_i, $i = 1, \ldots, N_a$. Endurance constraints and limits on resources cannot be directly addressed by this formulation. Timing constraints are addressed in Algorithm 2, guarantees that if the completion ordering satisfies order constraints, the resulting times of occurrence satisfy (3.22). If two tasks that have an order relationship between them are assigned to different agents, the timing must be enforced by the constraints (3.22).

To determine the cost of a feasible solution to the optimization problem the following must be known: the agents that are assigned to all of the tasks, the order in which each of the agents will complete the tasks it is assigned to, and the kinematics it will follow to perform these tasks. The set TA represents the assignments for each task and the partially ordered set $\mathcal{C}o$ represents the order of the task completion. The optimization procedure is performed in two levels. The upper level performs the optimization of the task assignments. The lower level performs the optimization of the completion order. The search procedure begins with an initial task assignment set. A 2-optimal completion order [73, 1] is constructed for this task assignment set and the cost of the resulting solution is evaluated. The task assignment set is altered systematically and a new 2-optimal completion order is constructed for each new set of assignments. This search proceeds, accepting new solutions that improve upon the best cost. The description of a solution to this problem contains all of the information concerning which agents are assigned to do which tasks and the order in which these tasks are to be completed. The vehicle kinematics determine precisely when the tasks will be completed. The nonlinearity in the objective function is due to the fact that the mission time is a function of only one agent's route. This route cannot be decided *a priori* as one must know the solution to know which route is the longest one. Hence the objective function is not a linear combination of the variables.

3.5 The Tabu/2-opt Heuristic Search

This section develops the Tabu/2-opt Algorithm that solves the VRP posed in Section 3.4. Algorithm 2 describes how a given task assignment and a corresponding completion order are used to determine the completion times of the tasks being serviced by the agents. This is followed by an evaluation of the algorithm's complexity.

Algorithm 2 begins with a solution and loops through the corresponding completion order. The algorithm first computes the maximum arrival time $\max_i T_{ji}$ of

any agent performing task t_{u_j}. It then checks the completion time of each of t_{u_j}'s precedents, $\mathcal{P}_{u_j}$. Note that in line 4 of Algorithm 2, the occurrence time of a task can be equal to that of its last occurring precedent task. The constraints of (3.22) can be enforced strictly by including an arbitrary, fixed period of time between a task and its last occurring precedent task. The start time of this task, $TS(t_{u_j})$, is then set to the maximum of these. Each agent performing t_{u_j} is assumed to arrive at t_{u_j} at $TS(t_{u_j})$.

Data: $TA, \mathcal{C}o$
1 **for** $j = 1$ **to** N_t **do**
2 $\quad T_{a_{\max}} = \max_i\{T_{u_j i}\}, \forall a_i \in TA(t_j)$
3 $\quad T_{\mathcal{P}_{\max}} = \max_k TS(t_k), \forall t_k \in \mathcal{P}_{u_j}$
4 $\quad TS(t_{u_j}) = \max\{T_{a_{\max}}, T_{\mathcal{P}_{\max}}\}$
5 $\quad T_{u_j i} = TS(t_{u_j}), \forall a_i \in TA(t_j)$
6 **end**
Result: TS

Algorithm 2: Timing evaluation

For each of the N_t tasks, the evaluations are as follows. Arrival time computation is $\mathcal{O}(N_a)$ in the worst case for checking every $a_i \in TA(t_j) = \mathcal{A}$. The worst case complexity for precedent occurrence time computation is $\mathcal{O}(N_t - 1) \simeq \mathcal{O}(N_t)$ for checking all tasks in $\mathcal{C}o$. Setting the actual start times is performed in $\mathcal{O}(N_t)$ time. The worst case complexity for the total evaluation is then $\mathcal{O}(N_a N_t^2)$. The cost of any solution given by TA and $\mathcal{C}o$ is $\max_j TS(t_j)$, $j = 1, \ldots, N_t$. This is evaluated in $\mathcal{O}(N_t)$ time. The cost of a solution is then evaluated in $\mathcal{O}(N_a N_t^3)$ time. This is easily reduced to $\mathcal{O}(N_a N_t^2)$ time if $\max_j TS(t_j)$ is evaluated during the timing evaluation. Thus cost evaluation for any solution to the optimization problem requires $\mathcal{O}(N_a N_t^2)$ time.

The following describes how the solution representation is helpful in guaranteeing that every solution considered obeys the assignment and order constraints imposed in the problem. These constraints are described in (3.22) and (3.24). These types of constraints appear naturally in cooperative control problems considering heteroge-

neous agents and complicated missions that contain order dependent outcomes [104]. The restrictions on which agents are permitted to be assigned to each task are captured in (3.24). The heuristic method will only choose assignments for tasks that are from this set. This ensures that all assignments obey the constraints in (3.24).

The next step in guaranteeing constraint satisfaction is to use the completion order set $\mathcal{C}o$ to describe the portion of the solution that represents the order in which tasks are performed. These partially ordered sets of tasks are constructed to guarantee order constraint satisfaction. The sets are then refined using an operation that preserves constraint satisfaction. Consider N_c order constraints of the form,

$$p_m = (TS(t_j), TS(t_k)), \forall m = 1, \ldots, N_c. \tag{3.25}$$

These order constraints (3.25) indicate that task t_j must be performed before task t_k. This input must be restructured to reflect the transitive relationships between tasks. For example, if task t_i must be performed before task t_j and t_j before task t_k, then t_i must also be performed before t_k. In this case, t_k must also be performed after t_i. The pairwise order constraints of (3.25) are restructured into the sets $\mathcal{P}_j$ and $\mathcal{D}_j$ of (3.17) and (3.18) for each task $t_j \in \mathcal{T}$.

The feasibility of the assignments is satisfied if at least one agent is selected to perform each task and all agents selected to perform each task $t_j \in \mathcal{T}$ belong to $Capability(t_j)$. This is possible provided the following is true:

$$Capability(t_j) \neq \emptyset, \quad \forall t_j \in \mathcal{T}, \tag{3.26}$$

$$TA(t_j) \neq \emptyset, \quad \forall t_j \in \mathcal{T}. \tag{3.27}$$

Precedence constraints can be input in the form of (3.25). These constraints must be linked together to form the sets $\mathcal{P}_j$ and $\mathcal{D}_j$ for each task. The feasibility of the

resulting constraints are checked by the following,

$$\mathcal{P}_j \cap \mathcal{D}_j = \emptyset, \quad \forall t_j \in \mathcal{T}. \tag{3.28}$$

The check in (3.28) says that no task t_j may have the restriction that there is a task t_k that must both come before and after the task t_j. For example, the constraints $p_1 = (TS(t_i), TS(t_j))$, $p_2 = (TS(t_j), TS(t_k))$, and $p_3 = (TS(t_k), TS(t_i))$ result in an infeasible instance of the problem.

The completion order is a partially ordered set. Order relations are defined between constrained tasks and tasks that belong to the same agent's route. Completion orders are constructed using a greedy method. Algorithm 3 details this process. The sets *available* and *notAvailable* refer to the sets of tasks that have had all of their precedent tasks selected and those that have not, respectively. The cardinality of a set is denoted by $|\cdot|$. Algorithm 3 is initialized with an empty completion order and all tasks are initially assumed unavailable. Each unavailable task is tested for availability by checking the completion order for its precedent tasks. If it has none or if the precedent tasks are included, it becomes available. Tasks are chosen from the set *available* according to which task will increase the final time the least. The procedure for determining available tasks is performed in $\mathcal{O}(N_t^2)$ time due to having to search $\mathcal{C}o$ for each task in *notAvailable*. The cost associated with adding each task $t_k \in$ *available* to the completion order is computed. The task that causes the minimum increase in cost is added to the completion order. Each cost function evaluation is done in $\mathcal{O}(N_a N_t^2)$ time. The entire evaluation takes $\mathcal{O}(N_a N_t^4)$ time. After the completion order is constructed for the set of task assignments, 2-opt refinement is used on the route.

The 2-opt exchange heuristic is a specific case of the variable k-opt heuristic at the heart of the Lin-Kernighan heuristic [73]. The 2-opt heuristic achieves in the

Data: TA
$\mathcal{C}o = \emptyset$
$notAvailable = \mathcal{T}$
$available = \emptyset$
while $\mid notAvailable \mid > 0$ **do**
  for $k = 1$ **to** $\mid notAvailable \mid$ **do**
    if $u_j \in \mathcal{C}o, \forall t_{u_j} \in \mathcal{P}_{u_k}$ **then**
      $available = available \bigcup \{t_{u_k}\}$
      $notAvailable = notAvailable \setminus \{t_{u_k}\}$
    end
  end
  $k_{min} = argmin_k cost(TA, \mathcal{C}o), t_k \in available$
  $\mathcal{C}o = (\mathcal{C}o, t_{k_{min}})$
  $notAvailable = notAvailable \setminus t_{k_{min}}$
  $available = available \setminus t_{k_{min}}$
end
Result: TA, $\mathcal{C}o$

Algorithm 3: Greedy completion order construction

neighborhood of 5% excess over the Held and Karp lower bound for single TSP instances using random Euclidean instances [1]. The basic 2-opt move cuts an agent route at two segments. Each segment is the link between two tasks or cities in a route. The problem is to find two of these segments (x_1 and x_2) that can be replaced with two different segments (y_1 and y_2) to reduce the cost of the route. This can be generalized to more than two cuts [73]. The version of the 2-opt exchange used here operates on the routes of the individual agents. The standard 2-opt exchange does not necessarily result in routes that obey order constraints. This feasibility is ensured by rejecting those routes that do not obey order constraints. A basic 2-opt move for a single closed tour (route whose start position is the same as its end position) is shown in Figure 3.1. The vehicle routes however do not necessarily begin and end at the same location. To accommodate this, the starting location and end location are connected by a link x_0 that is never considered as a candidate for the 2-opt exchange. This is shown in Figure 3.2. Note that for the basic 2-opt exchange, x_1 and x_2 each correspond to a pair of tasks. For any pair of cuts, x_1 and x_2, if the cuts are adjacent,

a route remains unchanged. All other pairs of x_1 and x_2 are tested for improvement.

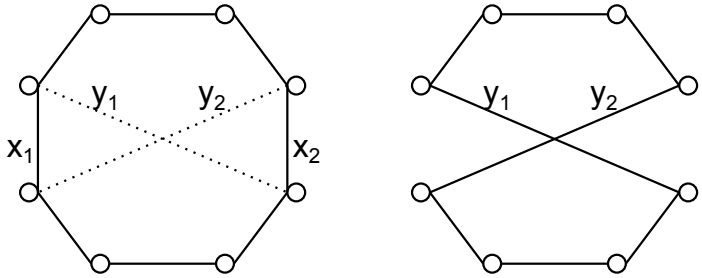


Figure 3.1: Standard 2-opt exchange on tours.

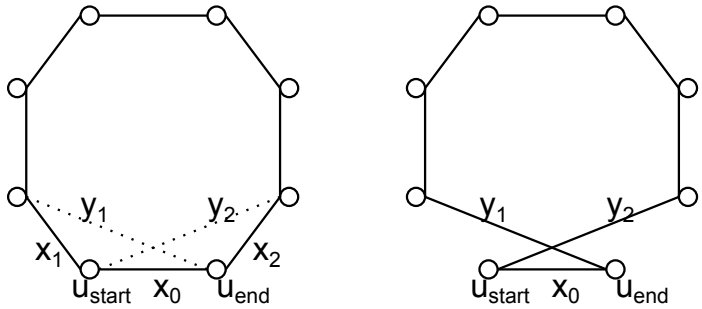


Figure 3.2: 2-opt exchange on vehicle routes.

The process of completion order improvement by 2-opt exchange proceeds as follows. Consider the completion order,

$$\mathcal{C}o = (t_1, t_2, t_3, t_4). \tag{3.29}$$

Let the route of agent a_1 be the totally ordered set, $(u_{start}, t_2, t_4, u_{end})$ and the single agent completion order be

$$\mathcal{C}o_1 = (t_2, t_4). \tag{3.30}$$

The only 2-opt exchange for this route results in

$$\mathcal{C}o_1 = (t_4, t_2). \tag{3.31}$$

The resulting agent route is then reinserted into the original completion order to

obtain

$$\mathcal{C}o = (t_1, t_4, t_3, t_2). \tag{3.32}$$

The resulting completion order is then tested for adherence to the precedence constraints. Resulting completion orders that violate precedence constraints are rejected and the process proceeds checking the remaining possible 2-opt exchanges until a feasible improvement is found or the possible 2-opt exchanges are exhausted. This process is performed for each vehicle's route; when an improvement for one vehicle's route is found, the improvement is kept and the process is repeated for the next vehicle's route. This process terminates when no further 2-opt improvements are possible for any route. The search through possible 2-opt exchanges requires $\mathcal{O}(N_t^2)$ time due to the worst case of searching each possible 2-opt exchange. The evaluation used to check if a new completion order violates precedence constraints requires $\mathcal{O}(N_t^3)$ computations. The timing evaluation of Algorithm 2 requires $\mathcal{O}(N_a N_t^4)$ computations. It can now be seen that if a particular problem instance is highly constrained, the 2-optimization of completion orders will be dominated by $\mathcal{O}(N_a N_t^5)$ computations instead of $\mathcal{O}(N_a N_t^6)$ computations. This is because the violating solutions are rejected before the timings are evaluated. For highly constrained problem instances, the algorithm spends most of its time rejecting solutions instead of checking costs. This is desirable because in practice, the algorithm runs faster when more constraints are introduced.

Tabu search is a metaheuristic used for combinatorial optimization [40]. The Tabu search can be viewed as searching a graph in the neighborhood of a given initial solution. The neighborhood of a task assignment set is as follows,

$$\mathcal{N}_{TA} = \{TA' \mid \exists! TA'(t_j) \text{ s.t. } TA(t_j) = TA'(t_j) \implies TA' = TA\}. \tag{3.33}$$

The Tabu search relies on a move to be defined that is used to perturb one solution

to another. The move used here is a perturbation in the assignment for each task. The feasible assignments $TA_f(t_j)$ for each task are enumerated as follows:

$$TA_f(t_j) = \{(t_j, \{a_1\}), (t_j, \{a_2\}), (t_j, \{a_1, a_2\}), (t_j, \{a_3\}), (t_j, \{a_1, a_3\}), \\ (t_j, \{a_2, a_3\}), (t_j, \{a_1, a_2, a_3\}), \ldots, (t_j, \{a_1, \ldots, a_{N_a}\})\}. \tag{3.34}$$

Each set $TA_f(t_j)$ represents one or more agents being assigned to a task. Let c_j be a bound placed on the number of agents allowed to perform task t_j. That is, a maximum of c_j agents will cooperatively perform task $t_j \in \mathcal{T}$. If $c_j = N_a$, the cardinality of each set of possible assignments to each task, $\mid TA_f(t_j) \mid$, grows exponentially with the number of agents. With c_j equal to a constant, $\mid TA_f(t_j) \mid$ is bounded by $\mathcal{O}(N_a^{c_j})$. The time to search the elements of $TA_f(t_j)$ is $\mathcal{O}(N_a^{c_j})$.

The set of feasible assignments is generated for each task only once at the start of the optimization. The Tabu search algorithm used in this work is summarized as follows. The termination criterion for the algorithm is reached when it has run a fixed number of iterations, $nIterations$. This number is based on a judgement of convergence by the user. The value used here was $nIterations = 100$. The Tabu search algorithm begins with an initial feasible task assignment TA_o. The neighboring task assignments $TA_j, j = 1, \ldots, \mid \mathcal{N}_{TA} \mid$ of the current task assignment TA correspond to task assignment sets in the neighborhood $\mathcal{N}_{TA}$ of the task assignment set for the current task assignment TA. The search then checks the neighboring solutions by searching the neighboring task assignment sets, generating the corresponding 2-optimal completion orders for them and checking the resulting solution for a cost improvement. The search through the neighborhood of a task assignment stores the best task assignment in that neighborhood as TA'. The function $cost$ evaluates the mission time for a given solution. If $cost(TA_j) < cost(TA_{best})$ the best solution known is replaced with TA_j. In the event that the search through the neighborhood of TA does not provide improvement, a local minimum has been reached and the current

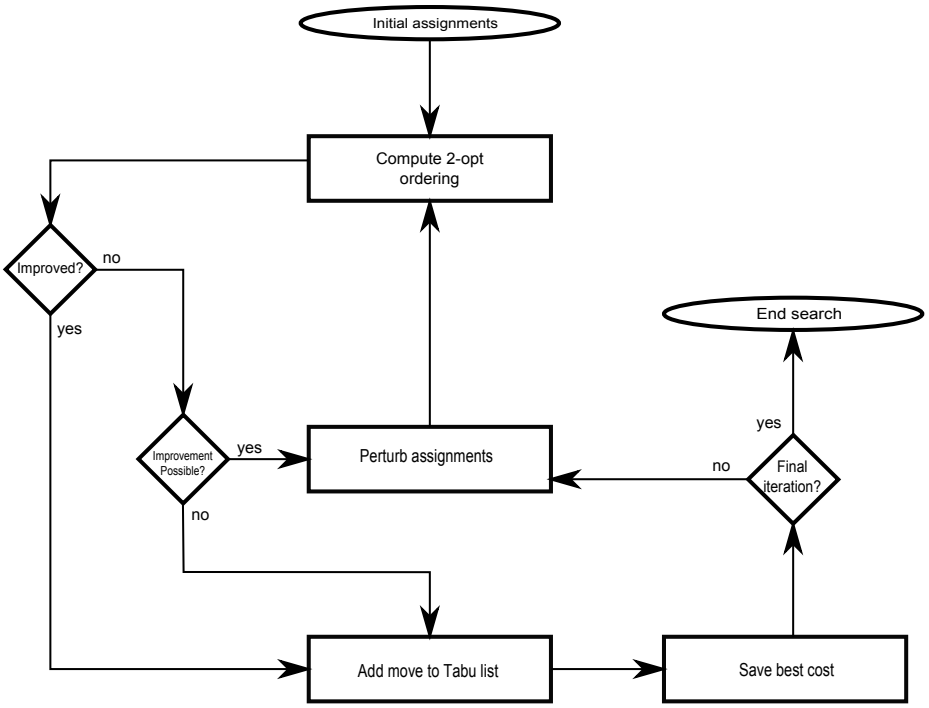


Figure 3.3: Overview of heuristic search.

solution is updated to the best neighboring solution found TA'. A Tabu list is kept, to prevent becoming stuck in a local minimum. When a new move is chosen, its inverse is added to the Tabu list. That is, a move that undoes that perturbation is not allowed while that move is on the Tabu list. A chart of this algorithm is presented in Figure 3.3.

The overall complexity at each step of the algorithm can be seen by combining our analysis of the completion order construction and refinement with the Tabu search procedure complexity. For each step of the Tabu search, the worst case number of perturbations in the assignments at the current solution will be $\mathcal{O}(N_a^{c_j} N_t)$ because this procedure could be performed for every $t_j \in \mathcal{T}$. An upper bound on the complexity at each step is the Tabu search step complexity combined with the complexity incurred by generating the refined completion orders at each step. This gives a step complexity of $\mathcal{O}(N_a^{c_j} N_t^7)$. For the following examples, $c_j = 1$, resulting in the algorithm scaling as $\mathcal{O}(N_a N_t^7)$. Physically, this corresponds to the case where agents do not cooperate

to perform individual tasks, but still coordinate task completion for tasks related by precedence constraints. With $\mid \mathcal{A} \mid \leq \mid \mathcal{T} \mid$, the step complexity is bounded by $\mathcal{O}(N_t^8)$.

In some cases, a Nearest Assignment heuristic is used to initialize the Tabu Search. This heuristic is simple and in many cases can provide an increase in solution quality. The Nearest Assignment heuristic initializes the task assignments as the lowest cost for each task as defined with respect to the initial configuration of the agents. This can be described as follows,

$$TA_{nn_j} = \min_{TA_f(t_j)} TS(t_j). \tag{3.35}$$

This heuristic physically manifests as choosing the agent that is capable of reaching the task the earliest. In the special case where all agents can travel at the same maximum speed, the heuristic has the effect of partitioning the tasks geographically. This clustering effect typically gives the assignment search a better initial starting point than a random initialization. The search then proceeds to satisfy all ordering constraints and reassigns tasks to improve cost. The Tabu/2-opt heuristic is built on methods that are traditionally effective for use on vehicle routing problems. However, it is still desired to compare the output of the heuristic to a method that gives optimal solutions to experimentally determine the typical optimality gap of the solutions.

A Branch and Bound method is used to search a tree for a solution with optimal cost. The Branch and Bound search algorithm is initiated by a Best First Search (BFS) [95] algorithm that provides an immediate feasible assignment. It starts with the root node of the tree and the estimated cost of each child is calculated by using the distance between the assigned agent and its assigned task. The child node with the smallest estimate is selected and the cost of the trajectory to perform the assignment is evaluated.

Several examples are chosen that can be treated by both the heuristic method

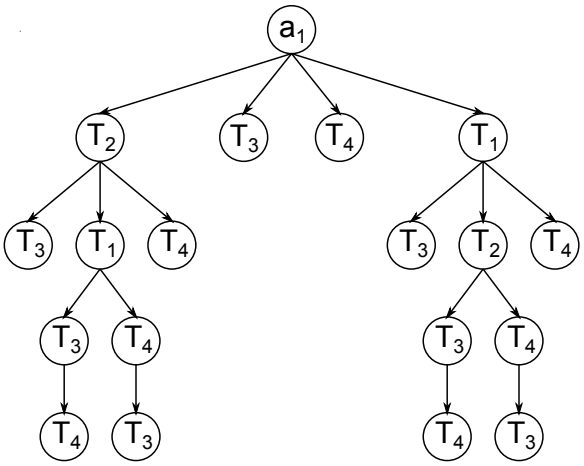


Figure 3.4: Tree structure.

and the optimal tree search. The computational limits for this implementation of the Branch and Bound search are reached at a problem size of about eight tasks and three agents. The agent initial positions lie on a 10×10 Euclidean grid and the agent maximum velocities are equal to 1 with consistent units. Tasks are to visit waypoints that lie on the 10×10 Euclidean grid. Figure 3.5 shows a problem instance.

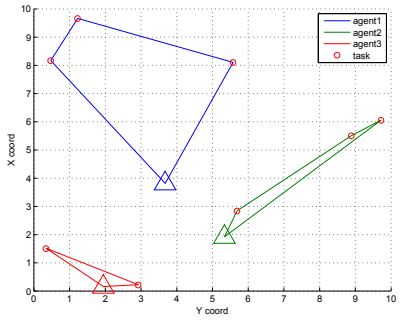


Figure 3.5: Problem instance for method comparison.

The results of mean, minimum, and maximum optimality gap for twenty randomly generated problem instances are presented are Table 3.1.

Table 3.1: Optimality Gap

Instance	**Mean**	**Min**	**Max**
Unconstrained	23%	6%	39%
Constrained	22%	9%	31%

Table 3.2: Completion times of constrained tasks.

Task	$TS(t_1)$	$TS(t_2)$	$TS(t_3)$	$TS(t_4)$	$TS(t_5)$	$TS(t_6)$
Completion time	11.1	14.5	25.7	13.0	16.1	16.9
Completing agent	a_3	a_2	a_2	a_1	a_2	a_1

Table 3.3: Completion times of constrained tasks.

Task	$TS(t_7)$	$TS(t_8)$	$TS(t_9)$	$TS(t_{10})$	$TS(t_{11})$	$TS(t_{12})$
Completion time	5.7	20.0	22.5	6.8	9.1	22.4
Completing agent	a_1	a_3	a_1	a_3	a_3	a_2

The optimality gap is defined as,

$$OG = \frac{cost_{heuristic} - cost_{opt}}{cost_{opt}}. \tag{3.36}$$

The solutions to the ten unconstrained cases show a mean optimality gap of 23% and the constrained examples show a corresponding optimality gap of 22%. For each agent, the user inputs the start and desired end position as well as the types of task that the agent is allowed to perform. For each task, the inputs are the beginning and final position for the task and the type of the task. In general, the agent may begin and end the task at different positions. These results are produced from instances of the problem that are subject to the following precedence constraints,

$$p_1 = (TS(t_1), TS(t_2)), \tag{3.37}$$

$$p_2 = (TS(t_2), TS(t_3)). \tag{3.38}$$

The following example contains simulation results for a scenario where three agents perform thirty tasks. In this example there are several precedence constraints that must be obeyed. This example demonstrates the performance of the search method on a problem similar to the multiple depot mTSP. The problem is for three agents to visit thirty tasks with the cost being the maximum time for any agent to finish its route and arrive at its final location. The final location is set to be its starting

location. The order constraints are,

$$p_1 = (TS(t_1), TS(t_2)), \tag{3.39}$$

$$p_2 = (TS(t_2), TS(t_3)), \tag{3.40}$$

$$p_3 = (TS(t_4), TS(t_5)), \tag{3.41}$$

$$p_4 = (TS(t_5), TS(t_6)), \tag{3.42}$$

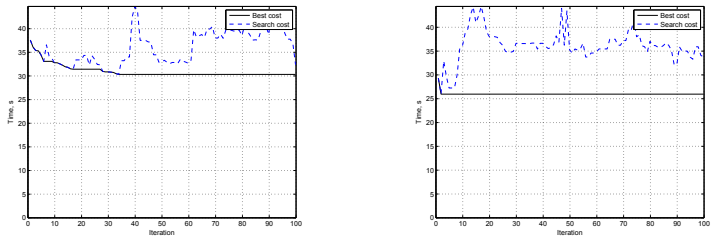
$$p_5 = (TS(t_7), TS(t_8)), \tag{3.43}$$

$$p_6 = (TS(t_8), TS(t_9)), \tag{3.44}$$

$$p_7 = (TS(t_{10}), TS(t_{11})), \tag{3.45}$$

$$p_8 = (TS(t_{11}), TS(t_{12})). \tag{3.46}$$

Here, solutions of the same problem instance are compared. First, the starting assignment for the optimization was chosen at random. Second, the starting assignment is chosen using the Nearest Assignment heuristic. Figure 3.6(a) and 3.6(b) show the convergence of the Tabu search for the two cases obtained after 100 iterations of the Tabu search. The dotted line is the current solution that is being perturbed throughout the optimization and the solid lower bounding line is the progress of the best solution.



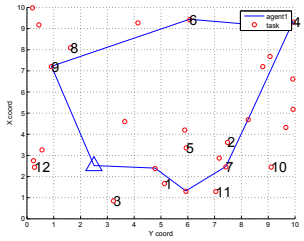
(a) Tabu search cost, random initial solution

(b) Tabu search cost, nearest initial assignment

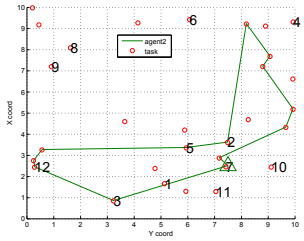
Figure 3.6: Tabu search cost.

Figures 3.7(a)-3.7(c) present the agent routes that result from a random initial

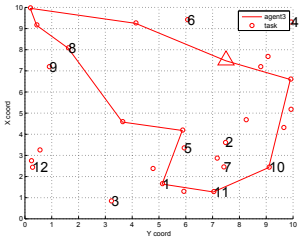
assignment. It was found that when the optimization algorithm is initialized with random initial assignments, the resulting solution tends to settle into equilibria where the agents loop around large portions of the plane. This seems to be due to the uniform nature of the initial assignments. These solutions tend to have gentle turns, although this is not explicitly enforced.



(a) Route of agent 1, random initial assignment.



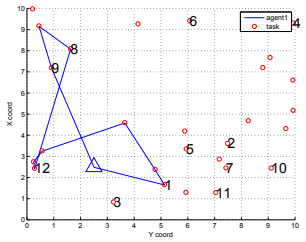
(b) Route of agent 2, random initial assignment.



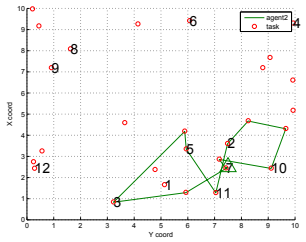
(c) Route of agent 3, random initial assignment.

Figure 3.7: Agent routes and results for random initial assignment.

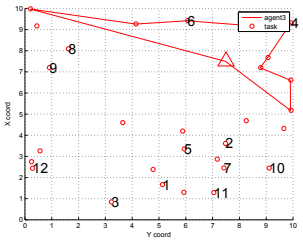
The times of completion of the order constrained tasks are listed in Table 3.2. Notice that even though several tasks are coupled through constraint relations, these tasks need not be serviced by the same agent. The algorithm is capable of enforcing the transitive order constraints without requiring those tasks to be done by the same agent. The routes resulting from the use of the nearest initial agent heuristic are presented below. These routes are more efficient with regards to moving shorter distances and to the global cost function itself. However, these routes tend to have many tight turns which are not penalized by the cost function or constraints. This can be seen by the plots of the routes in Figures 3.8(a)-3.8(c).



(a) Route of agent 1, nearest initial assignment.



(b) Route of agent 2, nearest initial assignment.

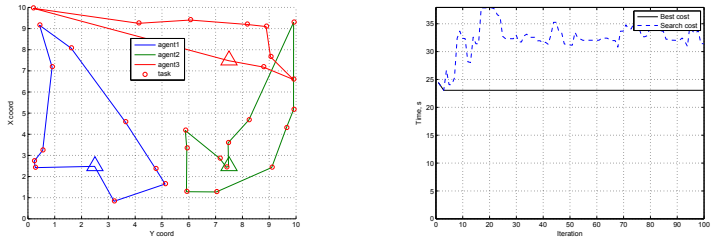


(c) Route of agent 3, nearest initial assignment.

Figure 3.8: Agent routes and results for nearest initial assignment.

The satisfaction of the constraints is independent of the type of initial assignment and all routes are 2-optimal with respect to the restrictions of the precedence constraints. It is worth noting that the routes of the agents cross each other. This occurs for the route of an individual agent and in an inter-agent sense. This is particular to the cost function used. For example, if the cost were the total length of all routes and not the mission time, this would not occur when the nearest agent heuristic is used or in optimal solutions. The precedence constraints also contribute to this phenomenon. In the presence of the precedence constraints, it can be beneficial for agents' routes to cross. Figure 3.9(a) shows an example with no precedence constraints. Notice that two of the agents routes cross due to the choice of cost function, but that because all individual routes are 2-optimal, no individual route crosses itself. This example also shows that the Nearest Assignment heuristic can be improved upon by the Tabu search. Figure 3.9(b) shows a slight increase in the solution quality from the initial assignment. These solutions are generated in approximately 18 seconds on a 2.4 GHz

MacBook.



(a) Routes with no precedence constraints.

(b) Corresponding cost.

Figure 3.9: Example solution with no precedence constraints.

It is interesting to understand what aspects of a problem, when changed or varied, cause a heuristic to deviate from optimality and by how much. That is, we would like to understand what causes a heuristic to perform worse. This understanding can help practitioners in determining the suitability of a heuristic for a given application. This understanding can be quantitative or qualitative.

The Tabu/2-opt heuristic employs a tabu list; this allows the heuristic to continue exploring candidate solutions after a local minimum is reached. The greedy nature of this heuristic enables it to rapidly improve the cost of the best known solution to a problem instance. The Tabu Search performs better than the purely greedy counterpart when the tabu list is "long enough" to ensure that the search can escape local minima. This judgement is problem dependent and the practitioner should tailor the tabu list length from experience.

The 2-opt repair heuristic is known to perform better for a smaller number of tasks than for a larger number of tasks [1]. This is the case for the k-opt heuristic in general. For agent routes visiting on the order of one hundred tasks the 2-opt heuristic provides low cost routes within 3% of optimality. The quality of the routes degrades as the number of tasks visited increases.

3.6 Measurement of Solution Quality

This section presents a measurement of solution quality for combinatorial problems. The task assignment and task scheduling problems addressed in this work are NP-hard. Techniques for overcoming this issue were discussed in Section 1.7. Metaheuristics such as [53, 88, 40, 61] can be used to solve a variety of problems involving objective functions and constraints that limit the effectiveness of analytical tools. In this section, we use a technique for measuring solution quality that relates the quality of a given solution to the distribution of solution quality for a problem instance. This technique only assumes that the objective function values take on finite values, the distribution of the objective function values only has one mode, and that feasible solutions can be readily generated (i.e., either directly or by rejection sampling). This is useful because it allows us to quantitatively measure the quality of a solution to a problem for which no analytical guarantees exist. The quality measurement is in the form of the probability of finding a better solution and quality relative to the space of possible solutions.

Consider the optimization problem

$$\min_{s \in \mathcal{S}} J(s) \tag{3.47}$$

s.t.

$$s \in \mathcal{C}_s \tag{3.48}$$

where $J : \mathcal{S} \to \mathcal{J} \subseteq \mathbb{R}$. The set $\mathcal{C}_s \subseteq \mathcal{S}$ represents the set of solutions that obey all constraints on the structure of the final solution. This work describes a way in which the set $\mathcal{J}$ can be statistically characterized. This characterization can in general be

non-Gaussian. In the majority of cases, this will not present a problem. A stochastic characterization of $\mathcal{J}$ allows us to measure the quality of solutions relative to the cost distribution of the entire solution space. The set $\mathcal{C}_s$ can be given by enumeration of all elements or by the properties of the elements. In the task assignment problem of section 3.5, the set $\mathcal{C}_s$ is defined using constraints on the characteristics of these elements.

The problem domain we consider has a useful stochastic structure. This structure is used to compute an estimate of the value of the cumulative distribution function corresponding to a solution's cost value. This gives a useful measure of the quality of a solution provided by an optimization algorithm. This method also gives perspective on the cost distribution of the problem domain. The cost distribution can in general be non-Gaussian (NG). This is overcome by using a method for approximating the cumulative distribution function (CDF) for any fractile on the non-Gaussian distribution by mapping that fractile to a Gaussian distribution. This method can be used for any single-modal distribution with finite first four moments.

The sample set is as defined in Eq. 3.48. Random solutions that satisfy all problem constraints are generated using a Las Vegas Algorithm, i.e., a random tree search. The key assumptions on the function J are that its image, $\mathcal{J}$, contains only finite values and that this mapping is deterministic. It is important that $\mathcal{J}$ contain only finite values to ensure finite statistical moments. The cost of these solutions is evaluated. These cost values represent outcomes of a random variable with sample space $\mathcal{J}$. They are independent and identically distributed (iid). It is the distribution of this random variable we seek to quantify.

The key descriptors for the distribution of cost values are the first four standardized moments of the unknown distribution. These moments are the mean, standard deviation, skewness and kurtosis. These moments are approximated by their corre-

sponding sample counterparts as follows.

$$\bar{J} = E[J(s)] \simeq \frac{1}{N}\sum_{i=1}^{N} J(s_i), \tag{3.49}$$

$$\sigma^2 = E[(J(s)-\bar{J})^2] \simeq \frac{1}{N}\sum_{i=1}^{N}(J(s_i)-\bar{J})^2, \tag{3.50}$$

$$\alpha_3 = \frac{E[(J(s)-\bar{J})^3]}{\sigma^3} \simeq \frac{1}{N\sigma^3}\sum_{i=1}^{N}(J(s_i)-\bar{J})^3, \tag{3.51}$$

$$\alpha_4 = \frac{E[(J(s)-\bar{J})^4]}{\sigma^4} \simeq \frac{1}{N\sigma^4}\sum_{i=1}^{N}(J(s_i)-\bar{J})^4. \tag{3.52}$$

Here E is the expectation operator, $s \in \mathcal{C}_s$ and N is the sample size. For convenience let the parameter $\tilde{J}_i = J(s_i) - \bar{J}$. This approximation is guaranteed to converge to the actual values of the respective statistical moments by the weak law of large numbers when all cost values are finite. In order to determine the convergence of the sample moments, we use a convergence criterion similar to the notion of settling time,

$$\max\left|\frac{1}{N}\sum_{i=1}^{N} J(s_i) - \frac{1}{k}\sum_{i=1}^{k} J(s_i)\right| < \delta, \tag{3.53}$$

$$\max\left|\frac{1}{N}\sum_{i=1}^{N} \tilde{J}^2 - \frac{1}{k}\sum_{i=1}^{k} \tilde{J}^2\right| < \delta, \tag{3.54}$$

$$\max\left|\frac{1}{N\sigma^3}\sum_{i=1}^{N} \tilde{J}^3 - \frac{1}{k\sigma^3}\sum_{i=1}^{k} \tilde{J}^3\right| < \delta, \tag{3.55}$$

$$\max\left|\frac{1}{N\sigma^4}\sum_{i=1}^{N} \tilde{J}^4 - \frac{1}{k\sigma^4}\sum_{i=1}^{k} \tilde{J}^4\right| < \delta, \tag{3.56}$$

$$k = N-1, \ldots, N-\Delta, N > \Delta. \tag{3.57}$$

This convergence criterion ensures that the deviations of the four sample moments is slower than δ of their current values in Δ samples. When this criterion is satisfied, the moments are considered to be converged and are then used to describe the distribution of $\mathcal{J}$. The use of σ in equations 3.55 and 3.56 refers to the current estimate of the

standard deviation.

The Hermite transformation [117] is used to transform fractiles from a non-Gaussian (NG) distribution to a Gaussian distribution. The Hermite transformation allows us to find the value of the CDF for any fractile on a NG distribution by transforming that fractile to a standardized Gaussian distribution and computing the value of the CDF in Gaussian space. The heart of the Hermite method uses a finite number of higher order statistical moments to create this transformation.

The idea is to approximate the outcome of a standardized NG process as a function of a standardized Gaussian random process. This function is a truncated power series function with terms that are the Hermite polynomials. The coefficients of the polynomials in this transformation are shown in Winterstein [117] to be functions of the higher order moments of the NG distribution. The effect is that the statistical moments of order higher than four are assumed negligible and only the first four statistical moments of the NG distribution are used in the transformation [117]. The transformation applied here is only able to treat single modal distributions. Let x and u be outcomes of random variables X and U where,

$$X = \frac{Y - \mu_Y}{\sigma_Y}, \tag{3.58}$$

$$U = \frac{V - \mu_V}{\sigma_V}. \tag{3.59}$$

The random variables Y and V correspond to non-standardized NG and non-standardized Gaussian processes respectively. The random variables X and U are the corresponding standardized random variables. The transformation from u to x and the corresponding inversions are developed in Winterstein [117]. The following transformation is the approximate mapping from the standardized NG response x to the standardized Gaussian response u.

This transformation differs depending on whether the response is softening or

hardening. A softening response refers to an NG distribution with a "fatter" tail than a Gaussian distribution. Such distributions have $\alpha_4 > 3$. A hardening response refers to a distribution with a "thinner" tail than a Gaussian distribution and $\alpha_4 < 3$. The measure of quality we seek is the CDF of the solution cost for the possibly NG distribution of $\mathcal{J}$. As such, we only require the transformation from x to u. The transformation for a softening response that maps from x to u is

$$u \simeq \xi(x)\left[(1+\psi(x))^3 + (1-\psi(x))^3\right] - a, \tag{3.60}$$

$$\xi(x) = (1.5b(a+x) - a^3)^{\frac{1}{3}}, \tag{3.61}$$

$$\psi(x) = \left[1 + \left(\frac{b-1-a^2}{\xi^2(x)}\right)^3\right]^{\frac{1}{2}}, \tag{3.62}$$

$$\tag{3.63}$$

where $a = \frac{h_3}{3h_4}$ and $b = \frac{1}{3h_4}$. The coefficients h_3 and h_4 are given by: $h_3 = \frac{\alpha_3}{3!}$ and $h_4 = \frac{\alpha_4 - 3}{4!}$. This transformation is monotone, that is the mapping is unique if the following criterion is satisfied,

$$h_3^2 < 3h_4(1 - 3h_4). \tag{3.64}$$

The transformation for a hardening response that maps from x to u is,

$$u \simeq x - h_3(x^2 - 1) + h_4(x^3 - 3x). \tag{3.65}$$

This transformation is monotone if the following criterion is satisfied,

$$16\alpha_3^2 < 9(3 - \alpha_3)(5 + \alpha_4). \tag{3.66}$$

We only consider distributions for which the relative skewness and kurtosis fall within the monotone limits. An extension to this is given in Choi [20] where non-monotone

transformations are considered. For all of these transformations, the skewness and kurtosis must be within the so called practical limit with relative values above the parabola,

$$\alpha_4 \geq \alpha_3^2 + 1. \tag{3.67}$$

3.6.1 A Traveling Salesman Problem Example

When the VRP of section 3.4 involves only one agent and no precedence constraints on tasks, the classical Traveling Salesman Problem (TSP) is recovered. We first use this well studied problem to illustrate our method for measuring solution quality. We use the implementation of the Lin-Kernigan Heuristic that is extensively detailed in Helsgaun [73]. This implementation has been shown to find optimal solutions for all known TSP instances that have been solved exactly. We will be concerned with a relatively small number of cities and so consider the results provided by this implementation as our baseline.

The TSP instances are generated by uniformly distributing city positions over a 10 by 10 unit square. All four moments converge after 21,519 samples. The values of these statistical moments for this example are $\bar{J} = 157$, $\sigma = 11.4$, $\alpha_3 = -0.11$, and $\alpha_4 = 2.95$. For this example, the values of the skewness and kurtosis deviate from corresponding Gausssian values of $skewness = 0$ and $kurtosis = 3$. This negative skewness causes the distribution to lean to the right. Figure 3.10 shows the convergence of the moments in this example.

Figure 3.11 displays a histogram of the cost data. The CDF of the optimal solution and that of the cost value found by the 2-opt heuristic both have numerical values of zero. These solutions lie at approximately -9.415σ from the mean.

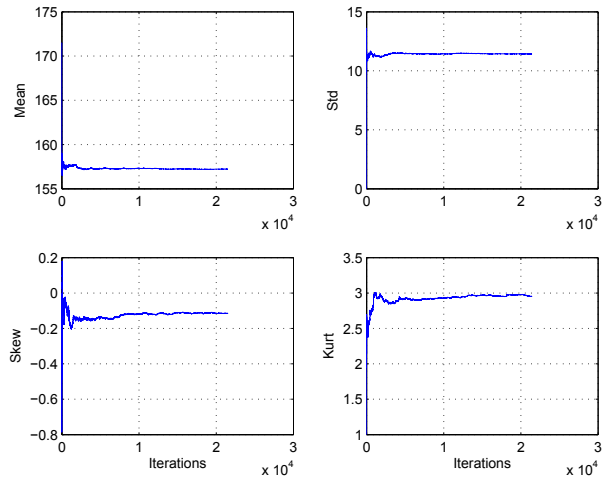


Figure 3.10: Convergence of first four moments for TSP example.

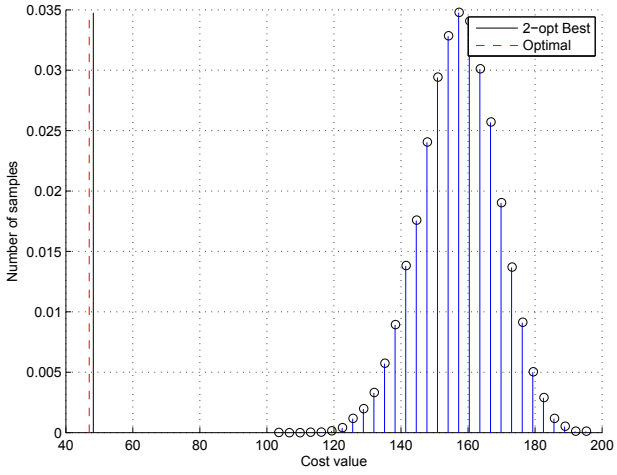


Figure 3.11: Histogram of cost data for TSP example.

3.6.2 Nonlinear Example and Results

This section demonstrates the solution of the full problem presented in Section 3.5. This problem is an example of a large scale nonlinear task assignment problem. The exact solution of this problem is not known to the best of the authors' knowledge.

We now consider the problem of Section 3.5 with two agents, all of which are able to complete all thirty tasks. This problem includes the following precedence constraints.

$$\mathcal{P}_1 = \emptyset, \quad \mathcal{D}_1 = \{t_2, t_3\} \tag{3.68}$$

$$\mathcal{P}_2 = \{t_1\}, \quad \mathcal{D}_2 = \{t_3\} \tag{3.69}$$

$$\mathcal{P}_3 = \{t_1, t_2\}, \quad \mathcal{D}_3 = \emptyset \tag{3.70}$$

$$\mathcal{P}_5 = \emptyset, \quad \mathcal{D}_5 = \{t_6, t_7\} \tag{3.71}$$

$$\mathcal{P}_6 = \{t_5\}, \quad \mathcal{D}_6 = \{t_7\} \tag{3.72}$$

$$\mathcal{P}_7 = \{t_5, t_6\}, \quad \mathcal{D}_7 = \emptyset \tag{3.73}$$

The cost function is $J(s) = \max_i T_{ei}$ and represents the final arrival time of the last agent to arrive at its final position. The cost distribution is shown in Figure 3.12. The statistical moments of $\mathcal{J}$ for this example converge to $\bar{J} = 97.02$, $\sigma = 11.28$, $\alpha_3 = 0.503$, and $\alpha_4 = 3.167$. We can see from the skewness and kurtosis that this distribution is NG. The histogram also clearly shows this fact. The best cost value $J^\star = 32.81 seconds$ has a CDF of approximately 1.8779×10^{-9} and a standardized value of -5.8946σ. The CDF was computed using the Hermite method and the standardized cost value is that of the corresponding Gaussian fractile.

Use of the Hermite method allows us to compute the CDF of the solution to the task assignment problem obtained from the Tabu Search. The CDF value and the

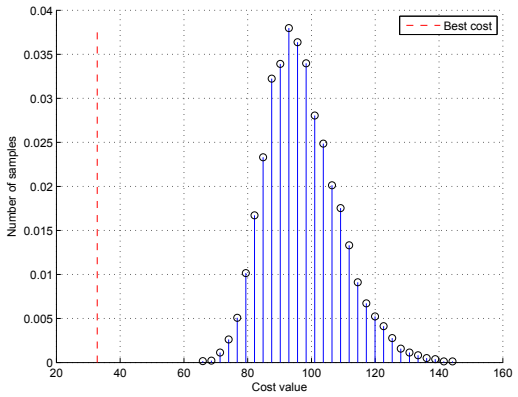


Figure 3.12: Histogram of cost data for Tabu/2-opt solution of multiple agent example.

σ location of this fractile indicate that this solution is of high quality. This measure ultimately uses random search as a benchmark. That is, the CDF value of a fractile is the probability of exceeding that value by generating a random solution from the Las Vegas method used. This method allows us to determine a relative and quantitative measure of solution quality by utilizing the generally non-Gaussian statistical characteristics of the cost function codomain in response to random sampling of feasible solutions.

The centralized task assignment and task scheduling problem formulation of this chapter provides a framework for minimizing mission time where a set of agents are assigned and scheduled to complete a set of tasks. The kinematic, capability, and precedence constraints are of interest to the vehicle routing and military mission planning community as detailed in Chapter II. The formulation of this chapter contributes a way to formulate a relevant class of mission optimization problems [104]. Instances of useful size, dozens of tasks and several agents, can be solved in reasonable time (seconds). Additionally, this chapter gives a technique to measure the quality of candidate solutions to problem instances relative to the set of possible solutions.

3.7 Limitations of Centralized Formulations

Centralized formulations allow us to express a multitude of hard problems. The vast literature on centralized task assignment and task scheduling algorithms gives us tools with which to solve these problems. In practice, the task assignment and task scheduling algorithm is run on a central computer. The agents that execute the task assignment and task schedule that is produced by the central computer must communicate with this computer to receive instructions. In situations where communication may be unreliable or incomplete, the centralization of task assignment and task scheduling threatens the ability to distribute the instructions to the executing agents. In situations where the computers may be unreliable, centralization threatens the completion of the task assignment and task scheduling process. These are fundamental reasons for considering distributed problem formulations that are able to model communication between agents and the failure of communication and agents. There are many additional benefits to developing distributed problem formulations and algorithms to solve them; these are discussed in Chapter IV.

CHAPTER IV

Distributed Task Assignment and Task Scheduling Problem Formulation

This chapter contributes a framework for distributed task assignment and task scheduling. This framework includes two problem statements: the first is the Communication-Constrained Distributed Assignment Problem (CDAP), the second is the Minimum-time Arbitrarily-constrained Distributed Scheduling Problem (MADSP). Solving these two problems results in: a task assignment that is feasible with respect to capability constraints and clustering constraints, and a minimum-time task schedule that satisfies scheduling constraints. The capability constraints are similar to those in Chapter III and the clustering constraints guarantee that two agents assigned tasks related by scheduling constraints can communicate. The scheduling constraints of the MADSP are predicates; these constraints can incorporate temporal and logical mission restrictions.

The separate treatment of these two problems allows distributed agents to find task assignments and task schedules that are feasible, optimal if possible, when using a communication network that has an arbitrary topology. The distributed algorithms presented in this chapter solve these problems. Correctness, completeness, and optimality is proven. Average and worst-case complexity results are detailed. The communication benefits of treating these problems separately are analyzed. The task

assignment and task scheduling algorithms must have specific properties to guarantee the correctness and completeness of the combined algorithm. These properties are discussed in detail.

The CDAP is important when assigning distributed agents to tasks where direct communication must be guaranteed between agents assigned to constrained tasks and the communication network topology is only locally known. Assuming only a connected network of arbitrary topology can limit the number of available communication links. The result is that the agents must find regions of the communication network that can satisfy capability and clustering requirements. The Stochastic Bidding Algorithm (SBA) is used by the agents to solve the CDAP. The correctness of the SBA is proven. The completeness of the SBA is analyzed, which indicates that the SBA will find a solution if one exists. The complexity analysis conservatively suggests average-case polynomial complexity. Auction methods are used for their efficiency; several of their core principles are used as the basis for the SBA. Unlike other auction methods, the SBA uses controlled randomness to find a global minimum of the relevant objective function. The randomness used here is reminiscent of Simulated Annealing. However, Simulated Annealing relies on centralized information about the solution and centralized authority to change the solution. In Simulated Annealing, the solution is manipulated using local search techniques. Knowledge of the entire solution and the ability to manipulate it must be centralized. Randomness is used in the SBA, but no agent ever has complete authority over the entire task assignment. Additionally, the distribution that characterizes the randomness is structured differently here.

In this chapter the dependence of the number of required communication links on the number of constraints coupling task assignment and task scheduling is quantified. The dependence of the number of required communication links on the generality of the problem description allows us to sacrifice problem expressiveness to operate on a

network with a reduced number of communication links. This reduced generality is modeled as a separation of task assignments and task schedules. The problem can be solved more efficiently if this coupling can be reduced.

The MADSP is important to task scheduling problems (such as described in Chapter I) where data describing tasks and constraints are distributed, the communication network topology may not be complete, and the communication topology is only locally known. The solution of this problem guarantees the simultaneous satisfaction of all mission constraints and the minimization of mission time. It is important that the constraints be satisfied with the minimization of the objective function. The Optimal Distributed Non-Sequential Backtracking Algorithm (OptDNSB) solves the MADSP. If a solution exists, the OptDNSB Algorithm is able to find a schedule that satisfies the arbitrary constraints while minimizing mission time. If no solution exists, all agents will recognize that there is no solution. The OptDNSB Algorithm is proven correct, complete, and optimal.

4.1 The Communication-Constrained Distributed Assignment Problem

This section develops the Communication-Constrained Distributed Assignment Problem (CDAP). The CDAP is to find a task assignment that obeys communication and capability constraints using the problem data that are distributed amongst several agents over a communication network that is unknown *a priori*. Here, the CDAP is converted into an optimization problem that, when solved, gives an assignment that obeys all constraints. The Stochastic Bidding Algorithm (SBA) solves this optimization problem by placing bids that are greater for agents whose assignments satisfy more constraints. The bids are adjusted stochastically to so that the algorithm avoids remaining stuck in local minima. The correctness of the SBA is proved and

the computational complexity analysis suggests polynomial average-case complexity with respect to key parameters.

The example scenario in Figure 4.1 is used to illustrate the concepts of this section. The example involves four unmanned aircraft, two of which have the capability to take pictures of targets, while the other two have the capability to attack targets. There are two unmanned ground vehicles that have the capability to track targets. There are two human operators that have the ability to confirm targets and authorize prosecution. The vehicles and operators are the agents in this example. The black lines in Figure 4.1 represent acknowledgement-based communication links. The prosecution of a target requires that the following tasks be performed: a photo must be taken of the target; an operator must confirm this target and authorize prosecution; confirmation must be relayed to the attacking aircraft; the attacking aircraft must attack the target; and the tracking vehicle must track the target while the latter is attacked. These tasks are defined formally in Section 4.1.1. In this scenario, there are four targets.

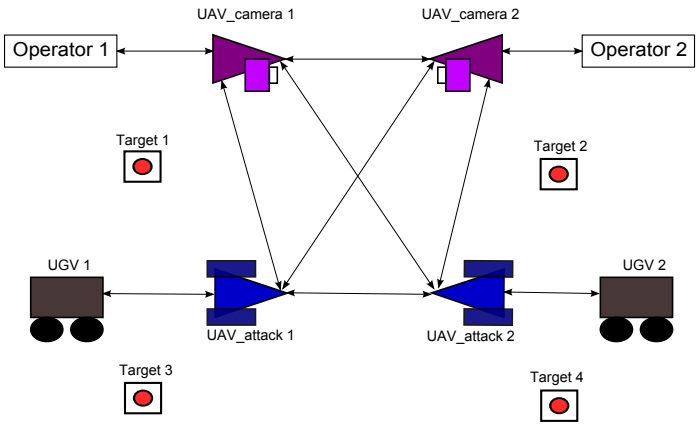


Figure 4.1: Military operations example.

Note that not all pairs of agents are connected by a communication link. Actual causes of such situations can result from the failure of individual communication links, a wireless network that has spotty or insufficient coverage, or the use of heterogeneous communication hardware or protocols. Agents know their own capabilities and those

of the agents they can communicate with, e.g., UAV_camera1 is capable of taking a picture. Agents know the constraints involving the tasks they are capable of performing, e.g., the agent that photographs target 1 must communicate with the agent that attacks target 1. Agents know the local topology of the communication network, e.g., UGV 1 does not know who UGV 2 can communicate with. These limitations motivate the statement of the CDAP and the development of distributed assignment algorithms that can solve this problem.

4.1.1 CDAP Problem Definition

This subsection details the concepts used to formulate the problem of this section. The concepts of this section are illustrated using the example scenario of Section 4.1. Only one target is considered for the sake of simplicity. The set of tasks to be assigned is as defined in 3.1. For the example of Section 4.1, $N_t = 5$ and the set of tasks is

$$\mathcal{T} = \{t_1, t_2, t_3, t_4, t_5\}, \tag{4.1}$$

where $t_1 \equiv$ photograph target 1; $t_2 \equiv$ classify target 1; $t_3 \equiv$ confirm status of target 1; $t_4 \equiv$ attack target 1; and $t_5 \equiv$ track target 1.

Tasks are to be assigned to agents. The set of agents is as defined in 3.2. For the example of Section 4.1, $N_a = 8$ and the set of agents is

$$\mathcal{A} = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, \}, \tag{4.2}$$

where $a_1, a_2 \equiv$ operator 1,2 respectively; $a_3, a_4 \equiv$ UAV_camera 1,2 respectively; $a_5, a_6 \equiv$ UAV_attack 1,2 respectively; and $a_7, a_8 \equiv$ UGV 1,2 respectively.

Here, a *task assignment* is a mapping from tasks to agents. It is defined formally as in 3.5. The following is an instance of a task assignment for the example of Section

4.1,

$$TA_1 = \{(t_1, a_3), (t_2, a_1), (t_3, a_4), (t_4, a_5), (t_5, a_8)\}. \tag{4.3}$$

Two notions of feasibility of task assignments are used here. The first refers to feasibility with respect to capability. The capability of the agents is described using the relation defined in 3.3. The capability relation for the example of Section 4.1, as it pertains to target 1, is

$$\begin{aligned} \mathit{Capability} = \{&(t_1, a_3), (t_1, a_4), (t_2, a_1), (t_2, a_2) \\ &(t_3, a_3), (t_3, a_4), (t_3, a_5), (t_3, a_6) \\ &(t_4, a_5), (t_4, a_6), (t_5, a_7), (t_5, a_8)\}. \end{aligned} \tag{4.4}$$

For the motivating example, the task assignment of (4.3) is feasible with respect to capability whereas the task assignment,

$$TA_2 = \{(t_1, a_1), (t_2, a_3), (t_3, a_2), (t_4, a_4), (t_5, a_4)\} \tag{4.5}$$

is not.

A central idea of this section is that tasks are bound to each other in the following sense. Tasks are related by operational constraints, as defined in (3.10), and the agents that are assigned such related tasks must be able to communicate in order to properly schedule for and perform these tasks. In the motivating example, tracking and attacking tasks for a single target are related by the operational constraint that they must be performed at the same time (i.e., when scheduled, these tasks must be scheduled to occur at the same time). Consider $N_s > 0$ task clusters,

$$\mathcal{T}_1, \ldots, \mathcal{T}_{N_s} \subseteq \mathcal{T}, \tag{4.6}$$

each of which represents a particular constraint and contains as elements the tasks

that are involved in each such constraint. For the example of Section 4.1 $N_s = 4$ and the clusters are

$$\mathcal{T}_1 = \{t_1, t_2\}, \tag{4.7}$$

$$\mathcal{T}_2 = \{t_2, t_3\}, \tag{4.8}$$

$$\mathcal{T}_3 = \{t_3, t_4\}, \tag{4.9}$$

$$\mathcal{T}_4 = \{t_4, t_5\}. \tag{4.10}$$

Define the *cluster group* relation between tasks,

$$CG \subseteq \mathcal{T} \times \mathcal{T}. \tag{4.11}$$

The cluster group relation is defined for a pair of tasks $(t_i, t_j) \in CG$ when there exists a sequence of intersecting clusters between t_i and t_j, and can be read t_i *is in the same cluster group as* t_j. By its definition, the cluster group relation is an equivalence relation, and therefore induces a partition of $\mathcal{T}$. The elements of this partition are referred to as cluster groups. In practice, cluster groups correspond to subsets of tasks that are not related by constraints. Figure 4.2 illustrates this for the motivating example.

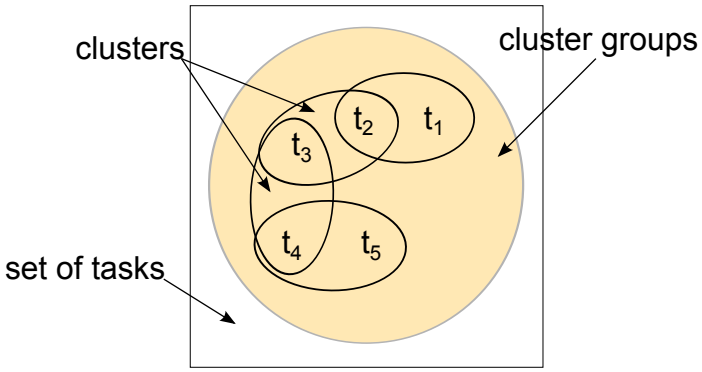


Figure 4.2: Cluster group for example scenario (target 1).

The agents in (3.2) have communication capability described by an undirected,

connected communication graph,

$$\mathcal{G}_c = (\mathcal{A}, \mathcal{E}_c). \tag{4.12}$$

There is an edge between two agents if and only if they are able to communicate directly with each other. The type of communication assumed here is acknowledgement-based, where each agent knows when communication is established with another agent. The edge set of the communication graph for the motivating example is

$$\begin{aligned} \mathcal{E} = \{&\{a_1, a_3\}, \{a_2, a_4\}, \\ &\{a_3, a_4\}, \{a_3, a_5\}, \{a_3, a_6\}, \\ &\{a_4, a_5\}, \{a_4, a_6\}, \\ &\{a_5, a_6\}, \{a_5, a_7\}, \{a_6, a_8\}\}. \end{aligned} \tag{4.13}$$

A task assignment is said to be *feasible with respect to clustering* if and only if

$$(\mathcal{A}, \mathcal{E}_c) \mid_{TA(\mathcal{T}_i)} \text{ is complete}, i = 1, \ldots, N_s. \tag{4.14}$$

Requirement (4.14) means that the agents that are assigned to the tasks belonging to a cluster must all be able to communicate directly with each other. The task assignment of (4.5) is feasible with respect to clustering, whereas the task assignment of (4.3) is not. The infeasibility of the task assignment TA_1 with respect to clustering results because agent $TA_1(t_2)$ cannot directly communicate with agent $TA_1(t_3)$ although t_2 and t_3 belong to the same cluster $\mathcal{T}_2$.

Definition IV.1. Feasible task assignment:
A task assignment that is feasible with respect to capability and feasible with respect to task clustering is said to be a *feasible* task assignment.

For the motivational example, the task assignment,

$$TA_3 = \{(t_1, a_3), (t_2, a_1), (t_3, a_3), (t_4, a_6), (t_5, a_8)\} \tag{4.15}$$

is feasible.

Proposition IV.2. *Let $X \in \{0,1\}^{N_t \times N_a}$ represent a task assignment so that $X_{ij} = 1$ if $TA(t_i) = a_j$ and $X_{ij} = 0$ otherwise. Then the problem of finding a feasible assignment can be formulated as a system of nonlinear equations in X.*

Proof. Proposition IV.2

Let matrix B_c represent the adjacency matrix of the graph $\mathcal{G}_c$, that is, $B_{c_{jl}} = 1$ if $\{a_j, a_l\} \in \mathcal{E}_c$, $B_{c_{jl}} = 0$ otherwise. Let the matrix B_{cl} represent the clustering relationships between tasks, that is, $B_{cl_{ik}} = 1$ if there exists a cluster $\mathcal{T}_m$ such that $t_i \in \mathcal{T}_m$ and $t_k \in \mathcal{T}_m$.

The matrix product $X^T B_{cl} X \in \{0,1\}^{N_a \times N_a}$ has the following meaning: $(X^T B_{cl} X)_{jl} = 1$ if the agents a_j and a_l are assigned tasks that share a cluster, $(X^T B_{cl} X)_{jl} = 0$ otherwise.

Let matrix AC represent the capability constraints. Here, $AC_{ij} = 0$ if $(t_i, a_j) \in$ *Capability* and 1 otherwise. The following constraints must be satisfied for the task assignment X to be feasible:

$$\begin{aligned} B_{c_{il}} - (X B_{cl} X^T)_{il} \geq 0, \\ i, l = 1, \ldots, N_a, \end{aligned} \tag{4.16}$$

$$\begin{aligned} \sum_{j=1}^{N_v} AC_{ij} X_{ij} = 0, \\ i = 1, \ldots, N_a, \end{aligned} \tag{4.17}$$

$$\sum_{i=1}^{N_a} X_{ij} = 1, \qquad j = 1, \ldots, N_t. \tag{4.18}$$

Equation (4.16) is nonlinear in the assignments, which proves Proposition IV.2. □

Proposition IV.2 illustrates the nonlinearity of the CDAP problem.

Each agent a_j is assumed to know the following data:

1. The set $Capability^{-1}(a_j)$, i.e., the tasks that a_j can perform,
2. The set $TA^{-1}(a_j)$, i.e., the tasks that a_j is assigned to,
3. $\forall t \in Capability^{-1}(a_j)$, $\mathcal{T}_m$ such that $t \in \mathcal{T}_m$, i.e., the clusters that contain the tasks in $Capability^{-1}(a_j)$,
4. The set $\mathcal{N}_{a_j}$, i.e., the neighborhood of a_j on the communication graph $\mathcal{G}_c$,
5. $\forall a_k \in \mathcal{N}_{a_j}$, the set $Capability^{-1}(a_k)$, i.e., the tasks that the neighbors of a_j can perform,

where $j = 1, \ldots, N_a$ and $m = 1, \ldots, N_s$.

The CDAP is for the agents in $\mathcal{A}$ to collectively find a feasible task assignment TA using only the available data together with communication with neighbors by (4.12).

The difficulty of this assignment problem is due to several factors: 1) the number of assignments that are feasible with respect to capability is $\mathcal{O}(\max_i(|Capability(t_i)|)^{N_t})$, $t_i \in \mathcal{T}$; 2) the clustering constraints are nonlinear; and 3) the problem data are distributed across a communication network of arbitrary topology.

With regards to assumptions, we have explicitly assumed that $\mathcal{G}_p$ is connected. The primary implication for a disconnected $\mathcal{G}_p$ is that assignments cannot be made

uniquely. That is, we cannot guarantee that a task assignment TA be a mapping. Our formulation is independent of the implementation of specific communication protocols used to guarantee acknowledgement-based communication. For instance, if multi-hop communication or ad-hoc networking is used to guarantee acknowledgement-based communication between two agents, this results in an edge between the two agents in $\mathcal{G}_c$. The associated effects on the network bandwidth are not addressed in this work.

4.1.2 Technical Approach

This subsection details the framework and the tools used to automate the solution of the assignment problem presented in Section 4.1.1. Define a set of messages $\mathcal{M}$, possibly infinite and closed under union. The content that the agents communicate to their neighbors originates in this set of messages. For every multi-index (i, j) such that $(t_i, a_j) \in Capability$, define a process similar to (1.12),

$$[t_i, a_j] = (States_{ij}, start_{ij}, trans_{ij}, msgs_{ij}). \tag{4.19}$$

This quadruple is called a *process*, where $States_{ij}$ is the state space of process $[t_i, a_j]$, i.e., a set of configuration quantities that may be boolean, integer, or real valued that describe the configuration of the process and represent its memory; $start_{ij} \in States_{ij}$ is the state at which process $[t_i, a_j]$ begins operation;

$$trans_{ij} \quad : \quad \mathcal{M} \times States_{ij} \to States_{ij}, \tag{4.20}$$

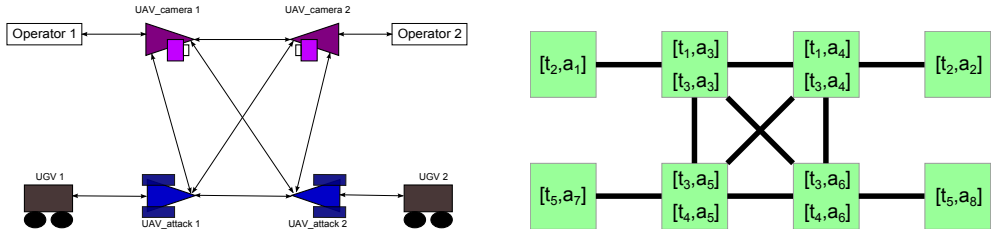
$$msgs_{ij} \quad : \quad \mathcal{M} \times States_{ij} \to \mathcal{M}. \tag{4.21}$$

Processes advance this state appropriately through the function $trans_{ij}$, which accepts incoming messages and produces a new state from the current state. The function $msgs_{ij}$ is responsible for reading received messages, the new state, and based on these, sending appropriate messages. Let $Processes$ be the set of processes defined

in (4.19). Define the undirected process graph $\mathcal{G}_p = (Processes, \mathcal{E}_p)$, where

$$\mathcal{E}_p = \{\{[t_i, a_j], [t_k, a_l]\} \mid \{a_j, a_l\} \in \mathcal{E}_c\}. \tag{4.22}$$

The process $[t_i, a_j]$ is connected to the process $[t_k, a_l]$ if agent a_j is connected to agent a_l by a communication link. The process graph for the motivating example is shown in Figure 4.3(b). The process graph is shown next to the communication graph of Figure 4.1 for illustration. The vertex set of the process graph for the motivating example (with one target) is given by (4.4) and the edge set follows from (4.13) and (4.22).



(a) (b)

Figure 4.3: Process graph for the example of Section 4.1.

The processes $[t_i, a_j] \in Processes$ form a distributed system in the sense specified in Section 4.1. The following modeling assumption is made here, the distributed system operates synchronously. That is, the processes each simultaneously update their state and then simultaneously send messages to their neighbors. Each iteration of computation and message transmission is referred to as a *round*.

The assumption of synchronicity allows us to reason about the operation of the system in discrete steps. In practice, the synchronous assumption can be relaxed to partially synchronous operation. It is possible to consider processes that compute (and communicate) at different rates, but wait to receive messages from their neighbors at each round. Message sending and reception must be reliable and messages must also be received in the order they are sent. In practice, we only need the system

to obey partial-synchrony. That is, processes may not operate at the same speed, but must wait to receive information from other bidding processes. This type of synchronization can result in rounds of unequal duration.

We describe a process' ability to satisfy assignment constraints locally as follows: we introduce the notion of a *cluster union*, defined for each task t_i as the set of tasks with which task t_i shares a cluster. Formally a cluster union is,

$$\begin{aligned} \mathcal{C}_i = \{t_k \mid \exists m \leq N_s : t_i \in \mathcal{T}_m \text{ and } t_k \in \mathcal{T}_m\}, \\ i = 1, \ldots, N_t. \end{aligned} \tag{4.23}$$

The cluster unions for each task in the example of Section 4.1 are

$$\mathcal{C}_1 = \{t_2\}, \tag{4.24}$$

$$\mathcal{C}_2 = \{t_1, t_3\}, \tag{4.25}$$

$$\mathcal{C}_3 = \{t_2, t_4\}, \tag{4.26}$$

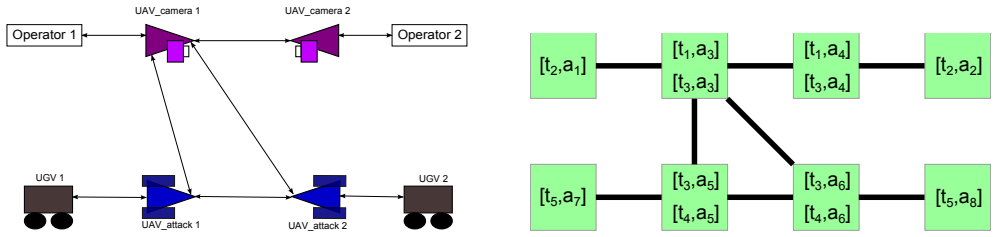
$$\mathcal{C}_4 = \{t_3, t_5\}, \tag{4.27}$$

$$\mathcal{C}_5 = \{t_4\}. \tag{4.28}$$

A process $[t_i, a_j]$ is *assignable*, informally, if its neighborhood contains processes whose agents can be assigned to the tasks $t_k \in \mathcal{C}_i$. Formally, we have the following,

Definition IV.3. Assignable Process:
Process $[t_i, a_j]$ is assignable if $\forall t_k \in \mathcal{C}_i$, $\exists [t_k, a_l] \in \mathcal{N}_{[i,j]}$.

Unassignable processes are not able to satisfy clustering constraints. To illustrate this concept, consider the modification of the example of Section 4.1 obtained by disabling the communication between a_4 and a_5 and between a_4 and a_6 as shown in Figure 4.4(a). It can be seen in Figure 4.4(b) that this results in process $[t_3, a_4]$ being unassignable.



(a) (b)

Figure 4.4: Modified example showing assignability and unassignability.

The problem of finding a feasible assignment is addressed using minimization. This is done by equating the satisfaction of the clustering constraints to the minimization of a carefully chosen objective function. The following discussion describes how this is done. For each $[t_i, a_j] \in Processes$, let

$$X_{ij} = \begin{cases} 1 & \text{if } TA(t_i) = a_j \\ 0 & \text{otherwise} \end{cases}, \tag{4.29}$$

specify whether or not t_i is assigned to a_j. For the example of Section 4.1 and the task assignment of (4.3), $X_{13} = X_{21} = X_{34} = X_{45} = X_{58} = 1$. The set of processes that allow process $[t_i, a_j]$ to satisfy the clustering constraints associated with task t_i is

$$\begin{aligned} \mathcal{NC}_{ij} = \{[k,l] \in \mathcal{N}_{[t_i,a_j]} \mid X_{kl} = 1 \\ \text{and } t_k \in \mathcal{C}_i\}, \end{aligned} \tag{4.30}$$

and has cardinality

$$nc_{ij} = \mid \mathcal{NC}_{ij} \mid . \tag{4.31}$$

Note that by the definitions of $\mathcal{NC}_{ij}$ and nc_{ij}, $0 \leq nc_{ij} \leq |\mathcal{C}_i|$. If $nc_{ij} = \mid \mathcal{C}_i \mid$ and $X_{ij} = 1$, then all required clustering constraints for task t_i are satisfied. For process $[t_1, a_3]$ and process $[t_1, a_4]$ and the task assignment of (4.3), $\mathcal{NC}_{13} = \{[t_2, a_1]\}$ and $\mathcal{NC}_{14} = \emptyset$. That is, $nc_{13} = 1$ and $nc_{14} = 0$.

Consider the set of all assignable processes whose agents belong to $Capability(t_i)$. This set is

$$\begin{aligned} \mathcal{B}_i = \{&[t_i, a_j] \in Processes \mid a_j \in Capability(t_i) \\ &\text{and } [t_i, a_j] \text{ is assignable}\}, \\ &i = 1, \ldots, N_t. \end{aligned} \tag{4.32}$$

For the example of Section 4.1 and task t_1, $\mathcal{B}_1 = \{[t_1, a_3], [t_1, a_4]\}$.

The *deficiency* of a process is defined as

$$nd_{ij} = \mid \mathcal{C}_i \mid - nc_{ij}. \tag{4.33}$$

Note that by (4.30), (4.31), and (4.33), $0 \leq nd_{ij} \leq |\mathcal{C}_i|$. The sum of this deficiency across the process graph is

$$J(TA) = \sum_{i,j:[t_i,a_j] \in Processes} nd_{ij} \cdot X_{ij}. \tag{4.34}$$

This objective function is never computed centrally and satisfies $J(TA) \geq 0$. The constraints of (4.14) are satisfied by minimizing the objective function $J(TA)$. Indeed, define the optimization problem

$$\min_{TA \in \mathcal{A}^{\mathcal{T}}} J(TA) \tag{4.35}$$

$$\text{s.t.} \quad TA \subseteq Capability.$$

The minimum value of any process deficiency, nd_{ij}, is zero and corresponds to $TA(t_i)$ being in communication with all agents $TA(t_k)$ where $t_k \in \mathcal{C}_i$, $i = 1, \ldots, N_t$.

Proposition IV.4. *$J(TA) = 0$ if and only if TA satisfies (4.14).*

Proof. Proposition IV.4

"$\rightarrow$" Assume that $\forall \mathcal{T}_m \subseteq \mathcal{T}$, $(\mathcal{A}, \mathcal{E}_c)|_{TA(\mathcal{T}_m)}$ is complete. This implies that $\forall \{t_i, t_k\}$ such that there is a $\mathcal{T}_m$ where $\{t_i, t_k\} \subseteq \mathcal{T}_m$, $\{TA(t_i), TA(t_k)\} \in \mathcal{E}_c$. This implies that $\forall [t_i, a_j] \in Processes$ where $X_{ij} = 1$, $nc_{ij} = |\mathcal{C}_i|$. By (4.34), $J(TA) = 0$.

"$\leftarrow$" Assume $J(TA) = 0$. This implies that $nc_{ij} = |\mathcal{C}_i|$ for all processes $[t_i, a_j]$ where $X_{ij} = 1$. Now, $nc_{ij} = |\mathcal{C}_i|$ for all $(t_i, TA(t_i)), i = 1, \ldots, N_t$, implies that for all $\{t_i, t_k\} \in \mathcal{T}_m, m = 1, \ldots, N_s$, $\{TA(t_i), TA(t_k)\} \in \mathcal{E}_c$. By (4.14), $\forall \mathcal{T}_m \subseteq \mathcal{T}$, $(\mathcal{A}, \mathcal{E}_c)|_{TA(\mathcal{T}_m)}$ is complete. □

Proposition IV.4 demonstrates that solving the optimization problem defined in (4.35) is equivalent to solving the CDAP. Note that nd_{ij} is a function of X_{kl}, where $[t_k, a_l] \in \mathcal{N}_{[t_i, a_j]}$. This introduces a nonlinearity into the objective function (4.34) that is similar to that of Proposition IV.2.

A pervasive concept in the auction literature is that of Pareto Optimality. Pareto Optimality is a property of an assignment of resources, tasks, etc. to agents. In the context of task assignment, a task assignment is Pareto Optimal (or efficient) if we cannot increase the total benefit of the task assignment by changing the assignment for any one task. Next we demonstrate this for the CDAP. Choose a benefit function for $a_j \in \mathcal{A}$,

$$b_j : \mathcal{A}^{\mathcal{T}} \rightarrow \mathbb{N} \tag{4.36}$$

that represents the benefit of an assignment TA to the agent a_j. The benefit function is

$$b_j(TA) = \sum_{i:(t_i, a_j) \in TA} nc_{ij}. \tag{4.37}$$

where nc_{ij} follows from 4.31. The total benefit of a task assignment TA is $\sum_{j:a_j \in \mathcal{A}} b_j(TA)$.

Proposition IV.5. *$J(TA) = 0$ if and only if TA is Pareto Optimal.*

Proof. Proposition IV.5

"$\rightarrow$" Let TA_1 and TA_2 both be Pareto Optimal. That is $\sum_{j:a_j \in \mathcal{A}} b_j(TA_1) = \sum_{j:a_j \in \mathcal{A}} b_j(TA_2)$. Consider $\sum_{j:a_j \in \mathcal{A}} b_j(TA_1)$, which attains its maximum when $nc_{ij} = |\mathcal{C}_i|$, i.e., $\sum_{j:a_j \in \mathcal{A}} b_j(TA_1)$ $= \sum_{i:t_i \in \mathcal{T}} |\mathcal{C}_i|$. From this, and 4.34, $J(TA_1) = \sum_{(i,j) \in TA_1} (|\mathcal{C}_i| - |\mathcal{C}_i|)$, $J(TA_1) = J(TA_2) = 0$.

"$\leftarrow$" Consider TA_1 and TA_2, let $J(TA_1) = J(TA_2) = 0$. This implies that $nc_{ij} = |\mathcal{C}_i|$, $(i,j) \in TA_1$. Similarly for TA_2. Hence, $\sum_{j:a_j \in \mathcal{A}} b_j(TA_1) = \sum_{j:a_j \in \mathcal{A}} b_j(TA_2) = \sum_{i:t_i \in \mathcal{T}} |\mathcal{C}_i|$. It follows that TA_1 and TA_2 are both Pareto Optimal.

□

4.1.3 Solution Procedure

This section discusses the SBA that is based on the principles of Simulated Annealing and uses controlled randomness to find a solution to the CDAP. Processes use the SBA presented in [52] to bid on behalf of their agents for the tasks that the corresponding agent is capable of performing. The Stochastic Bidding Procedure (SBP) is run by each process. The SBA is the aggregate of all SBPs running on all processes, together with a termination condition.

4.1.4 The Stochastic Bidding Procedure

Two types of messages are used, *bid* messages and *next* messages. A *bid* message contains the numerical bid by a process for its respective task. A *next* message tells that the sending process has received all of the bids it was expecting. When a process bidding for a task has received *bid* messages from all processes bidding for the same task, it determines the winner as the process with the highest bid. This allows the

processes to determine the winner for themselves without relying on an auctioneer. The bids are computed such that it is unlikely (with probability equal zero) that two bids be identical. Thus there is a unique winner and each bidding process computes the same winner (with probability equal one).

The bids are computed as follows. Let the quantity ND_{ij} be a local estimate of the value of $J(TA)$ in (4.34). Processes $[t_i, a_j] \in Processes$ update the value of ND_{ij} every time *next* messages are received from processes $[t_k, a_l]$ where $X_{kl} = 1$. Let q_{ij} be a random variable with probability density function

$$pdf(q_{ij}) = \begin{cases} \frac{1}{2\sqrt{2\pi\sigma_{ij}^2}} exp[-\frac{q_{ij}^2}{\sigma_{ij}^2}], q_{ij} > 0, & \text{for } \sigma_{ij} > 0, \\ 0, & \text{for } \sigma = 0 \end{cases}, \tag{4.38}$$

with standard deviation,

$$\sigma_{ij} = \frac{ND_{ij} \cdot c}{T}, \tag{4.39}$$

where the parameter c is constant and T goes to infinity as time elapses. The bid values for each assignable process $[t_i, a_j] \in Processes$ are computed as,

$$bid_{ij} = nc_{ij} - q_{ij}. \tag{4.40}$$

The bids computed in (4.40) favor processes that satisfy their respective clustering requirements dictated by $\mathcal{T}_m$, $m = 1, \ldots, N_s$.

Tasks are bid on in round-robin order. Without loss of generality, the bidding begins with task t_1. Bidding begins for $t_i \in \mathcal{T}$ when a process $[t_i, a_j] \in \mathcal{B}_i$, has received *next* messages from every $[t_{(i-1)}, a_j] \in \mathcal{B}_{i-1}$, where t_0 is defined as t_{N_t} by round-robin. When a process bidding for t_i has received *bid* messages from every $[t_i, a_j] \in \mathcal{B}_i$, that process computes the winning bidder and sends a *next* message. Note that each process maintains local authority over the task it bids on and when

it bids.

These messages are relayed by each process across the graph $\mathcal{G}_p$. A round refers to a process running *trans* and *msgs*; and a *session* refers to the completion of bidding for each of the N_t tasks, that is, one turn of round-robin. In (4.39), T is equal to the number of sessions. This procedure can be thought of as a distributed Simulated Annealing method. The bidding procedure is depicted graphically in Figure 4.5.

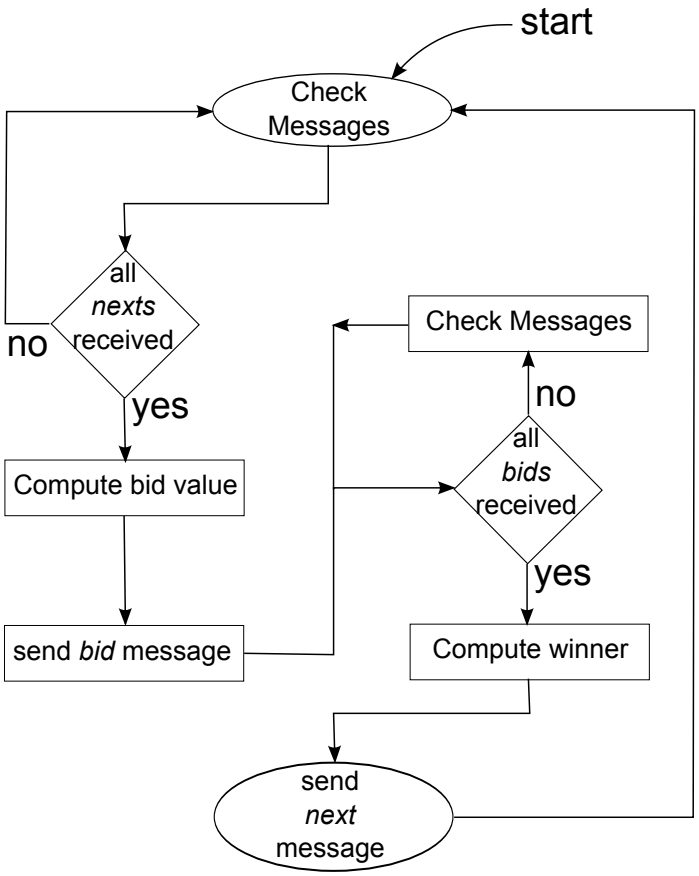


Figure 4.5: Bidding procedure diagram.

The SBP is described formally as follows. The state $state_{ij}$ of process $[t_i, a_j] \in$

Processes and the messages, $\mathcal{M}$ are defined as follows,

$$\begin{aligned} state_{ij} = (j, i, \mid \mathcal{C}_i \mid, nc_{ij}, \\ ND_{ij}, N_{sent_{ij}}, bid_{ij}, X_{ij}, \\ allBidRecvd_{ij}, allNextRecvd_{ij}, \\ sendBid_{ij}, sendNext_{ij}), \end{aligned} \tag{4.41}$$

and

$$\begin{aligned} \mathcal{M}_{bid} = \{(N_{sent_{ij}}, j, i, \\ bid_{ij}, \mid \mathcal{C}_i \mid, nc_{ij})\}, \end{aligned} \tag{4.42}$$

$$\begin{aligned} \mathcal{M}_{next} = \{(N_{sent_{ij}}, j, i, \\ X_{ij}, \mid \mathcal{C}_i \mid, nc_{ij})\}, \end{aligned} \tag{4.43}$$

$$\mathcal{M} = \mathcal{M}_{bid} \cup \mathcal{M}_{next}. \tag{4.44}$$

The quantities $\mid \mathcal{C}_i \mid$, nc_{ij}, ND_{ij} and bid_{ij} are computed per their definitions. The quantity $N_{sent_{ij}}$ is the number of messages sent by process $[t_i, a_j]$. The boolean quantities X_{ij}, $allBidRecvd_{ij}$, $sendNext_{ij}$, and $sendBid_{ij}$ are initialized as zero, and the boolean quantity,

$$allNextRecvd_{ij} = \begin{cases} 1 & \text{if } i = 1 \\ 0 & \text{otherwise} \end{cases}. \tag{4.45}$$

Define the function *computeBid*, which computes a bid value by (4.38), (4.39), and (4.40). Also define the function *computeX*, which determines if process $[t_i, a_j]$ is the winning bidder for task t_i. This is done after all bids for task t_i are received by process $[t_i, a_j] \in Processes$. The function $forwardNew$ sends all new incoming messages to all neighbors except for the sending process, and *sendMsg* sends a message $M \in \mathcal{M}$ to all neighbors. Algorithms 4 and 5 detail the operation of the $trans_{ij}$ and $msgs_{ij}$

functions respectively for the algorithm presented in this section.

Data: $M, State_{ij}$
1 **if** $allNextRecvd$ **then**
2 | $allNextRecvd = 0$
3 | $computeBid(ND_{ij}, nc_{ij})$
4 | $sendBid = 1$
5 **end**
6 **if** $allBidRecvd$ **then**
7 | $allBidRecvd = 0$
8 | $computeX()$
9 | $sendNext = 1$
10 **end**
Result: $state_{ij} \in states_{ij}$

Algorithm 4: $trans_{ij}$

Data: $state_{ij}$
1 $forwardNew()$
2 **if** $sendBid$ **then**
3 | $sendBid = 0$
4 | $M = M_{bid}$
5 | $sendMsg(M)$
6 **end**
7 **if** $sendNext$ **then**
8 | $sendNext = 0$
9 | $M = M_{next}$
10 | $sendMsg(M)$
11 **end**
Result: $M \in \mathcal{M}$

Algorithm 5: $msgs_{ij}$

4.1.5 Liveness Condition

Note that the procedure depicted in the diagram in Figure 4.5 does not terminate. The liveness condition for this algorithm is specified as follows:

Livenesss condition:

$\forall [t_i, a_j] \in Processes$, $ND_{ij} = 0$.

This liveness condition is a function of the states of each process and thus requires current knowledge of each process which, in general, no process will have. Rather than terminate, it is enough that there exists a round at which the processes collectively output a solution. In the case of the Stochastic Bidding procedure there is a round at which the processes output a feasible assignment. In addition, after this round, the processes output this assignment and only this assignment. The liveness condition relies on knowledge of the states of each process.

There are several properties of the SBA worth noting. As $J(TA)$ and similarly ND_{ij} decrease, the probability that a process that satisfies a large number of its communication requirements wins the bidding for its task increases. As a result, the bid that any process $[t_i, a_j] \in Processes$ can place for its task is maximized when $\sigma_{ij} = 0$. This corresponds to $ND_{ij} = 0$, which implies that every process that has won the bidding for its task can communicate with all processes that have won the bidding for the tasks in $\mathcal{C}_i$. This implies that if process $[t_i, a_l]$ can match process $[t_i, a_j]$'s bid for task t_i, then process $[t_i, a_l]$ can also meet the same communication requirements as process $[t_i, a_j]$.

It would be realistic to have a constraint that clustered tasks be assigned to distinct agents regardless of capability. This constraint can be incorporated by restricting the set $\mathcal{B}_j$ to vary with the assignment. That is, if two tasks must be assigned to separate agents, said agents should not bid on both tasks.

The number of rounds required for the sharing of all *bid* messages is upper bounded by $diam(\mathcal{G}_p)$. The number of rounds required for all of the processes bidding on the next task to receive all *next* messages is also upper bounded by $diam(\mathcal{G}_p)$. This results in a upper bound between the beginning of bidding for t_i and t_{i+1} of $2 \cdot diam(\mathcal{G}_p)$.

4.1.6 Analysis of Stochastic Bidding

This section presents an analysis of the SBA. The SBA developed above is able to find the global minimum of the objective function in (4.34). This is contrary to other auction methods that, while they perform well, may not find a global minimum when minimizing a nonlinear objective function. The SBA is unique in that it uses controlled randomness to achieve this. While the SBA has a Simulated Annealing character, it is distinctly different in that it is a distributed algorithm where no agent has global authority over the solution. It is important to show that the SBA converges to a task assignment that is feasible. That is, it should converge and it should converge to a solution to the CDAP. This section shows the SBA to be correct and to find a solution if one exists. The correctness is as follows,

Proposition IV.6. *Correctness:*
The Stochastic Bidding Algorithm terminates if and only if a feasible assignment TA has been found.

Proof. Proposition IV.6
"$\rightarrow$" Assume bidding has terminated, that is $ND_{ij} = 0$ for all processes. Now, $ND_{ij} = 0$ for all processes implies that for all processes, $nd_{ij} \cdot X_{ij} = 0$. By the definition of nd_{ij} and X_{ij}, and Proposition IV.4, TA is feasible.

"$\leftarrow$" Assume TA is feasible. By the definition of a feasible task assignment and Proposition IV.4, $J(TA) = 0$. By the connected graph assumption, each process will have $ND_{ij} = 0$ within $diam(\mathcal{G}_p)$ rounds. Moreover, $ND_{ij} = 0$ for all processes implies liveness. □

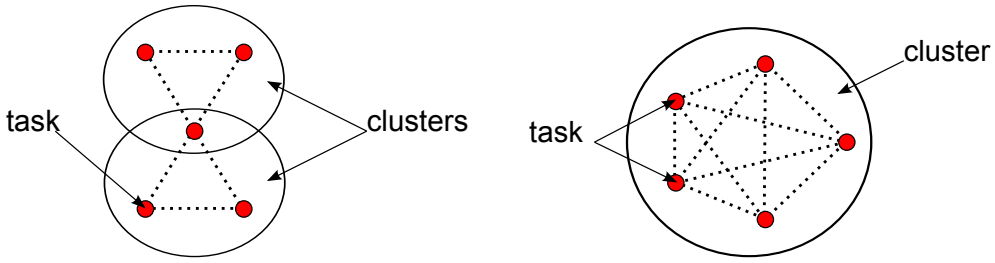
Proposition IV.6 concerns the ability of the SBA to recognize that a solution has been

found and terminate.

Here we discuss the complexity of the SBA. Since each process corresponds to one task, the cluster groups also induce a partition on the set of processes. This partition is composed of the subsets of processes that bid for tasks in each cluster group. Without loss of generality, we can consider the assignment of a single cluster group because processes bidding within different cluster groups will never be related by constraints.

The following discussion concerns the analysis of the complexity of the SBA. For this analysis, it is assumed that each task has the same number of agents capable of performing it. That is, $|Capability(t_1)| = \ldots = |Capability(t_{N_t})| = N$. The number of assignments that are feasible with respect to capability is $\mathcal{O}(N^{N_t})$. This motivates the study of the effects of N and N_t on computational complexity.

Define the *connection index* CI to be the number of inter-process connections required by clustering constraints. Given a set of tasks and the corresponding clusters, it is simple to determine CI. The maximum value of $CI = \frac{N_t(N_t-1)}{2}$ and the minimum value of $CI = N_t - 1$ and are referred to as CI_{max} and CI_{min} respectively. For the example of Section 4.1, $CI = 4 = CI_{min}$. Figures 4.6(a) and 4.6(b) illustrate five tasks and two different cluster configurations. The dotted lines illustrate the communication requirements that will be imposed between the agents that are assigned to the tasks. The configuration of Figure 4.6(a) presents a clustering configuration that corresponds to $CI = 6$ and Figure 4.6(b) presents a clustering configuration that corresponds to $CI = 10$. The quantity CI is additionally illustrated by comparing it with $\mathcal{NC}_{ij}$ in (4.30) and $J(TA)$ in (4.34). The maximum cardinality of $\mathcal{NC}_{ij} =\mid \mathcal{C} \mid$, notice that if this is the case for all processes $[t_i, a_j]$ where $X_{ij} = 1$ then $\sum_{ij} nc_{ij} = 2CI$ and $J(TA) = 0$. It follows that the maximum that $J(TA)$ can attain is $2CI$. Therefore by choosing CI, we are directly choosing the difficulty of the problem as it relates to the effects of clustering.



(a) CI = 6 (b) CI = 10

Figure 4.6: Tasks are related by clusters. Clusters impose communication requirements.

To investigate the computational complexity of the SBA, we investigate the number of sessions needed to find a feasible assignment as N, N_t, and CI change. The parameter c in (4.39) is important to the convergence of the bidding procedure. Its effect is analyzed separately.

The importance of a session relates to the fact that it represents the change of the current tasks assignment across the network with respect to each task. By measuring the number of sessions taken to find a feasible solution, we are able to abstract the structure of the graph in studying the algorithm's complexity. That is, the relative location (on $\mathcal{G}_p$) of agents that are bidding on a single task affects the number of rounds needed to bid on that task. However, the number of sessions required to bid on all tasks once is one regardless of the structure of $\mathcal{G}_p$.

We study the computational complexity of the SBA experimentally. Note that the parameters N and N_t fix the number of processes at $|Processes| = N \cdot N_t$. Recall that the process graph $\mathcal{G}_p$ must satisfy the connectedness assumption. Also, it is only necessary to consider processes that are assignable. It is important that there exist a solution. Stochastic Bidding cannot output that there is no feasible assignment even if there does not exist one. This is a result of using stochastic search. The following procedure is used to construct $\mathcal{G}_p$:

1. Pick N, N_t, CI

2. Create $N \cdot N_t$ processes and N_t tasks

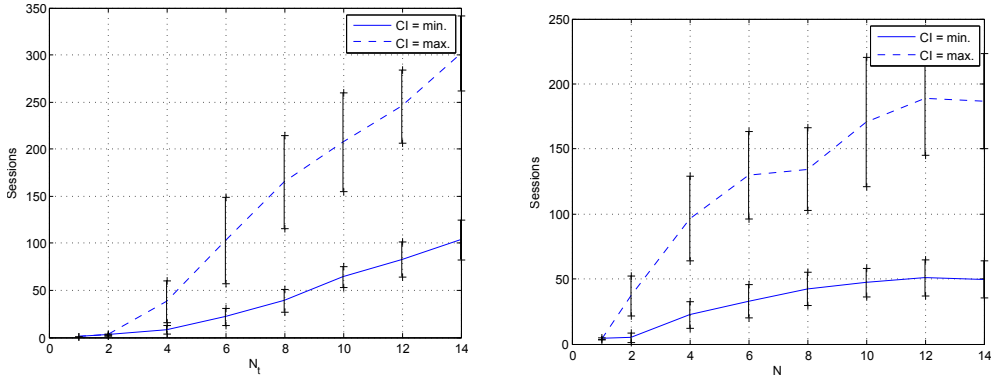
3. Create the cluster group as follows. Create a graph with tasks as vertices. Pick two tasks at random and connect them by an edge; this is the cluster group. Pick a task that is in the cluster group and one that is not in the cluster group at random and connect them. Repeat this until all tasks are in the cluster group. Select pairs of tasks that are not connected by an edge at random and connect them by an edge until the number of edges is equal to CI.

4. Create $\mathcal{G}_p$ as follows. Pick two processes at random and connect them by an edge. Select at random a process that is connected to the others and one that is not and connect them. Repeat this until $\mathcal{G}_p$ is connected. Pick at random one process corresponding to each task. Connect them according to the cluster group.

5. For each process, if it is not assignable, select at random processes that will allow it to meet the assignability criterion and connect them.

This analysis considers instances of $\mathcal{G}_p$ that are created to satisfy the necessary assumptions. These instances of $\mathcal{G}_p$ are not created using $\mathcal{G}_c$ from (4.12). Several phenomena motivate this type of analysis. Recall that $\mathcal{G}_p$ represents the capabilities of the agents and how those capabilities are related by communication links. This abstraction is useful because it lets us focus only on the capabilities of the agents and how those capabilities are related by communication links. Note that when considering $\mathcal{G}_c$ in (4.12), the addition of communication links in $\mathcal{G}_c$ represents the addition of one or more communication links in $\mathcal{G}_p$. Satisfying the requirements of the existence of a solution, connectedness of $\mathcal{G}_c$, and the assignability requirements for processes implies that these requirements are satisfied for $\mathcal{G}_p$. Thus performing the complexity analysis for random instances of $\mathcal{G}_p$ results in conservative estimates of algorithm complexity.

The results in Figures 4.7(a) and 4.7(b) describe the complexity behavior of the SBA. Each line in Figure 4.7(a) was created using data points collected at two task intervals along the abscissa, similarly for N in Figure 4.7(b). Notice that from the plots, this spacing is sufficient to suggest the nature of the mapping from N_t to *sessions* and similarly for N to *sessions*. Each of these data points represents the mean of 30 randomly generated experiments created by the above procedure. The spread lines show the one standard deviation bounds for the resulting number of sessions required to find a feasible assignment for each collection of experiments.



(a) N = 8 (b) Nt = 8

Figure 4.7: Number of sessions needed to find a feasible assignment.

Figure 4.7(a) shows how the mean number of sessions required to find a feasible assignment using the SBA changes as the number of tasks is increased. Here, the value of N is kept constant and $N = 8$. Figure 4.7(a) contains the results for $CI = CI_{max}$ and $CI = CI_{min}$. Figure 4.7(b) shows how the mean number of sessions required to find a feasible assignment changes as N is increased. Here, the value of N_t is kept constant at $N_t = 8$. Figure 4.7(b) contains the results for $CI = CI_{max}$ and $CI = CI_{min}$.

The main drivers of computational complexity are N_t and CI. Increasing the value of CI clearly increases the number of sessions needed to find a feasible task assignment. However, the shapes of the curves plotted for CI_{min} and CI_{max} are qualitatively similar. Additionally, the spread of the data is consistently larger for

the CI_{max} curves. In practice this means that we will be less confident about the number of sessions needed to find a feasible assignment when CI is large. The results of Figure 4.7(a) show that the effect of increased agent capability (i.e., increased N) on the computational complexity of the algorithm is less than linear. That is, for a fixed CI, there appears to be a linear upper bound on computational complexity as a function of N. This suggests that while increasing the number of capable agents increases the time to find a solution, the algorithm becomes less sensitive to this effect as N is increased. The results of Figure 4.7(b), while not as benign as those of Figure 4.7(a), show a distinctly linear behavior of the computational complexity as a function of N_t.

The relative behavior of the plots in Figures 4.7(a) and 4.7(b) are also important. Recall that an upper bound on the number of possible task assignments is $\mathcal{O}(N^{N_t})$, under the assumption that $\forall t_i \in \mathcal{T}$, $|Capability(t_i)| = N$. This statement says that the effect of N_t on the computational complexity can be exponential whereas the effect of N is polynomial. The computational experiments show a linear expected computational complexity and that the qualitative difference in the effects of N_t and N on computational complexity remain. It is also interesting to note that the linear complexity with respect to the number of tasks is seen in the classic auction algorithm [7].

For every experiment performed, there is a value of c in (4.39) for which the SBA finds a solution. The following demonstrates that the effect of an increase in c is to increase the uniformity of the bidding procedure. That is, increasing c makes agents more equal with regards to the competitiveness of their bids. From (4.39) we can see that

$$\lim_{c \to \infty} \sigma_{ij} = \infty. \tag{4.46}$$

It follows that

$$\lim_{c \to \infty} pdf(q_{ij}) = U(0, \infty), \tag{4.47}$$

where $U(a, b)$ is the uniform distribution with support (a, b). From this it follows that as c goes to infinity, bid_{ij} has the same distribution for each process. The effect of c is to increase the number of sessions T required to bias bidding toward processes that satisfy clustering constraints. Equation (4.47) implies that during any session, there is a non-zero probability of generating a feasible assignment. This is important because it tells us that c is a tuning value for the SBA that allows us to insert randomness into the algorithm to further explore the solution space.

4.1.7 The Dependence of Required Communication Links on Modeling

The coupling between task assignment and task scheduling is addressed here. The number of communication links required for task scheduling depends on how the problem of finding a task assignment and a task schedule is modeled. The mechanism for this dependence is that the methods of [49, 121, 120] require that agents assigned to tasks related by constraints be able to communicate. For a communication network of arbitrary topology, these communication links may not be available. The SBA in this section is able to find assignments that guarantee that this requirement is met while only requiring that the communication network be connected. Scheduling can then be done using [49]. This section motivates a way of modeling the task assignment and scheduling problem using constraints that separate the task assignment and schedule. The results of this section show that modeling the problem in this way reduces the number of required communication links needed for finding a task assignment and task schedule.

A task schedule is defined as in 3.7). Informally, scheduling *constraints* are used to characterize the allowable schedules. Scheduling constraints are formally defined as functions that map from the set of schedules to the set $\{false, true\}$. The number of constraints is N_s. These constraints are defined as in (3.10).

A generalized specification of the problem of finding a task assignment and task

schedule can be given using constraints of the form (3.8). Constraints such as these affect not only the times at which tasks are performed, but also the agents that perform them. Using this type of constraint imposes the additional communication requirements that result in the dependence detailed as follows.

Consider that the set of tasks $\mathcal{T}$ can be redefined as follows. Let $\mathcal{T}'$ be a new set of tasks with elements

$$t_i^j \in \mathcal{T}' \text{ s.t. } (t_i, a_j) \in Capability. \tag{4.48}$$

This represents the task t_i being performed by agent a_j. Consider task assignments that operate on the new set of tasks,

$$TA' : \mathcal{T}' \to \mathcal{A}. \tag{4.49}$$

The capability relation becomes

$$Capability' = \{(t_i^j, a_j) \mid (t_i, a_j) \in Capability\}, \tag{4.50}$$

Note that there is only one task assignment that is feasible with respect to $Capability'$,

$$TA' = \{(t_i^j, a_j) \mid (t_i, a_j) \in Capability\}. \tag{4.51}$$

This results from the fact that each of the tasks $t_i^j \in \mathcal{T}'$ corresponds to the specific agent $a_j \in \mathcal{A}$.

Consider constraints similar to those in (3.10) that operate over the set of schedules for $\mathcal{T}'$,

$$p'_m : \mathbb{T}_s^{\mathcal{T}'} \to \{false, true\}, \tag{4.52}$$

and the resulting clusters $\mathcal{T}'_m \subseteq \mathcal{T}'$, $m = 1, \ldots, N'_s$. The constraints of (4.52) are

equivalent to those in (3.8). That is, the constraints of (4.52) incorporate all possible assignments of tasks to agents. Note that in addition to the constraints of (3.10), the following constraints are required to guarantee that TA is a mapping,

$$p'_i = \begin{cases} 1 & \text{if } \exists! j : TS(t_i^j) > 0, j = 1, \ldots, N_a \\ 0 & \text{otherwise} \end{cases}, \tag{4.53}$$

where $i = 1, \ldots, N_t$, hence $N_s \leq N'_s$. The constraints of (4.53) ensure that for each $t_i \in \mathcal{T}$, t_i is performed by only one agent. This representation incurs the following penalty. The constraints in (4.53) result in the clusters $\mathcal{T}'_i$. The task assignment TA' is the only task assignment that is feasible with respect to $Capability'$. For TA' to be feasible with respect to clustering,

$$(\mathcal{A}, \mathcal{E}_c)|_{TA'(\mathcal{T}'_m)}, \tag{4.54}$$

must be complete, $m = 1, \ldots, N'_s$. This requirement imposes the additional communication constraints associated with the clusters $\mathcal{T}'_i$, $i = 1, \ldots, N_t$. Physically, this requirement means that the agents in the set $Capability(t_i)$ must be able to communicate directly. For the example of Section 4.1, this requirement results in the need for the additional communication links in Figure 4.8.

Let $N_i =\mid Capability(t_i) \mid$. The cost of this modeling decision is that

$$L = \sum_{i=1}^{N_t} \frac{N_i(N_i - 1)}{2}, \tag{4.55}$$

additional communication links must be available.

For the following analysis assume $N_1 = \ldots = N_{N_t} = N$, that is, each task has an equal number of capable agents. Under this assumption, the number of additional communication links required in Equation 4.55 is $L = N_t \frac{N(N-1)}{2}$. The number of

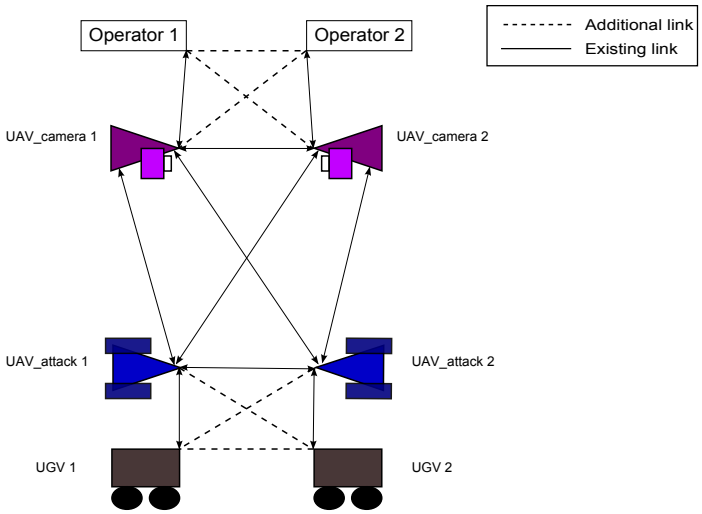


Figure 4.8: Illustration of additional communication links required by the generalized model.

additional communication links is plotted in Figure 4.9 as a function of N. Hence,

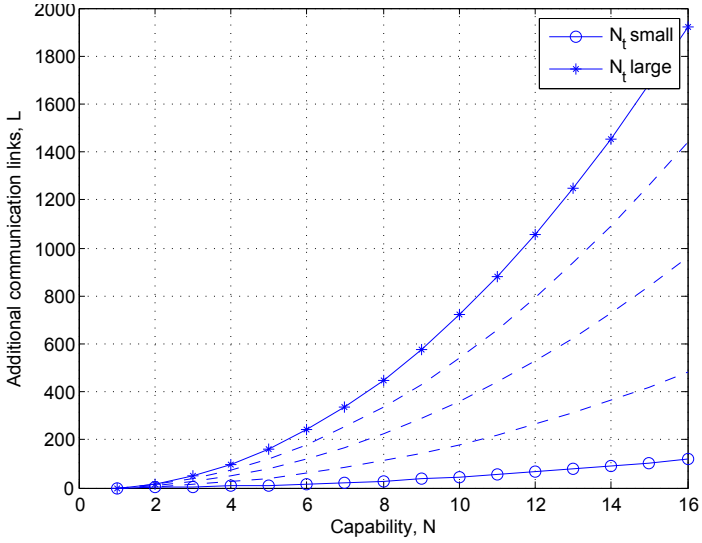


Figure 4.9: The number of additional communication links required between processes if the assignment and schedule are not separated, plotted as a function of the capability of the agents.

the quantification of the task assignment and task schedule coupling is given by (4.55) and Figure 4.9.

Solving the CDAP gives a task assignment that satisfies communication constraints. Satisfaction of these constraints guarantees that the agents assigned to tasks

that are involved in scheduling constraints can communicate directly. The specification of the Minimum-time Arbitrarily-constrained Distributed Scheduling Problem of the following section includes communication assumptions that the CDAP satisfies. The Optimal Distributed Non-Sequential Backtracking Algorithm exploits the fact that the SBA satisfies these assumptions. The relationship between the satisfaction of these communication assumptions by an algorithm such as the SBA and the scheduling algorithm is discussed in detail in the next section.

4.2 Distributed Constrained Minimum-Time Schedules in Networks of Arbitrary Topology

This section presents the details of the Minimum-time, Arbitrarily-constrained, Distributed Scheduling Problem (MADSP). This is the problem of finding a minimum-time schedule subject to arbitrary constraints using problem data that are distributed amongst several agents over a communication network topology that is only locally known. The Optimal Distributed Non-Sequential Backtracking (OptDNSB) Algorithm solves the MADSP. This section presents proofs of the correctness, completeness, and optimality of the algorithm. It is shown that the OptDNSB Algorithm retains these properties under conditions where the task assignment changes during scheduling. This is important because it admits a class of task assignment algorithms, that includes the SBA, to be used that can satisfy the communication constraints necessary for scheduling using the OptDNSB Algorithm.

The example scenario in Figure 4.10 is used throughout this section. This is a fire fighting example considering two houses, each surrounded by three fires. The goal is to extinguish at least two of the three fires surrounding each house and rescue the people inside as quickly as possible using three vehicles. The tasks are: the extinguishing of the six fires (F1-F6) and the rescue of the people in the two houses (H1-H2). These tasks are constrained in the following way: at least two of the three fires near each house must be extinguished, this must be done before the rescue, and each vehicle can only perform one task at a time. The tasks are defined formally in Section 4.1.1. The vehicles are two aircraft (EV1 and EV2) and one ground vehicle (R1). The two aircraft are able to extinguish fires. The ground vehicle is able to rescue the people from each house. These three vehicles are referred to more generally as agents.

The black lines between EV1 and R1, and R1 and EV2 in Figure 4.10 represent reliable communication links. Note that not all pairs of agents are connected by a

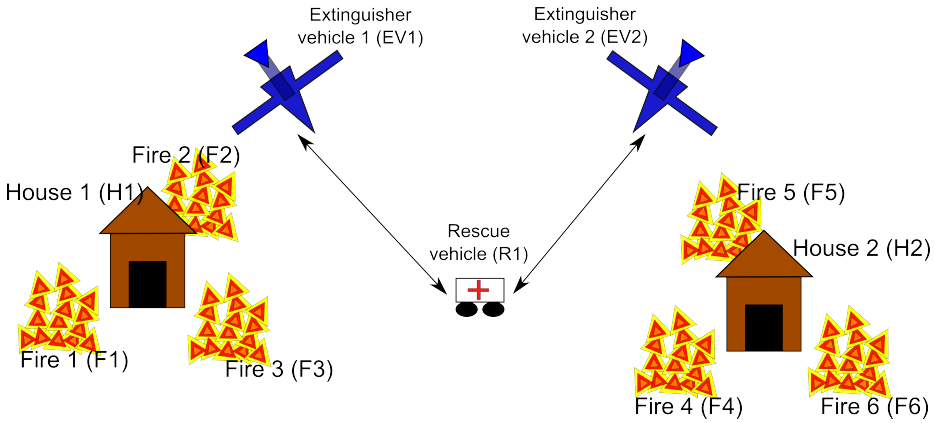


Figure 4.10: Cooperative rescue example.

communication link. Such situations can result from the failure of individual communication links, a wireless network that has spotty or insufficient coverage, or the use of heterogeneous communication hardware or protocols. Agents know their own capability and that of the agents they can communicate with, e.g., EV1 is capable of extinguishing a fire. Agents know the constraints involving the tasks they are capable of performing, e.g., at least two of the three fires around each house must be extinguished. Agents only know the local topology of the communication network, e.g., EV1 knows that it can communicate with R1, but does not know with whom R1 can communicate.

In this example, the assignment of tasks to agents has been made *a priori*. The agents must schedule the tasks while obeying the aforementioned constraints on task completion order and choice. The tasks must be scheduled to minimize the time needed to perform all required tasks. The need to do minimum-time scheduling under these limitations motivates the statement of the MADSP and the development of distributed scheduling algorithms that can solve this problem.

4.2.1 MADSP Problem Definition

This section details the concepts that are used to formulate the MADSP. The concepts of this section are illustrated using the example scenario of Section 4.2.

The set of tasks to be assigned is as defined in (3.1). For the example of Section 4.2 $N_t = 8$ and the set of tasks is

$$\begin{aligned}\mathcal{T} = \{&t_1, t_2, t_3, t_4, \\ &t_5, t_6, t_7, t_8\},\end{aligned} \tag{4.56}$$

where $t_1, t_5 \equiv$ rescue H1 and H2; $t_2, t_3, t_4 \equiv$ extinguish F1, F2, and F3; and $t_6, t_7, t_8 \equiv$ extinguish F4, F5, and F6.

Tasks are assigned to agents. The set of agents is as defined in (3.2). For the example of Section 4.2, $N_a = 3$ and the set of agents is

$$\mathcal{A} = \{a_1, a_2, a_3\}, \tag{4.57}$$

where $a_1, a_3 \equiv$ EV1 and EV2, and $a_2 \equiv$ R1.

Here, a task assignment is a mapping and is defined as in (3.5). The following is an example task assignment for the example of Section 4.2:

$$\begin{aligned}TA = \{&(t_1, a_2), t_2, a_1), (t_3, a_1), (t_4, a_1), \\ &(t_5, a_2), (t_6, a_3), (t_7, a_3), (t_8, a_3)\}.\end{aligned} \tag{4.58}$$

The agents in (3.2) have communication capability described by an undirected, connected communication graph in (4.12). There is an edge between two agents if and only if they are able to communicate directly with each other. The type of communication assumed here is acknowledgement-based, where each agent knows when communication is established with another agent. The edge set of the communication graph for the motivating example is:

$$\mathcal{E} = \{\{a_1, a_2\}, \{a_2, a_3\}\}. \tag{4.59}$$

A *duration* function is used to describe the length of time required to complete a task. The duration of a task is defined as a function that maps from $\mathcal{T}$ to a finite subset of the set of integers $\mathbb{T}_s \subseteq \mathbb{N}$,

$$D : \mathcal{T} \to \mathbb{T}_s. \tag{4.60}$$

Tasks are assumed to have known duration. In many practical situations, the duration of a task may depend on a number of factors: the agent performing the task; the order of task completion; and the choice of which tasks to complete. It is possible to model the problem in a more general way where, in addition to the stop time (or start time) of the task, the duration is chosen and must satisfy constraints. These modeling decisions affect how the scheduling method must be designed to guarantee completeness. Here, the duration of a task is only a function of the task. An example duration function for the example of Section 4.2 is

$$\begin{aligned} D = \{&(t_1, 1), (t_2, 1), (t_3, 1), (t_4, 1), \\ &(t_5, 1), (t_6, 1), (t_7, 1), (t_8, 1)\}. \end{aligned} \tag{4.61}$$

A task *schedule* is a mapping and is defined in (3.7). Here, $\mathbb{T}_s = \{0, \ldots, s\}$ and $s \leq s_{max}$, the *scheduling horizon*, is an integer that is adjusted iteratively. The integer s_{max} is an upper bound on the duration of the mission that is known *a priori*. The set $\mathbb{T}_s$ where $s = s_{max}$ is $\mathbb{T}_{max}$. We refer to the elements of $\mathbb{T}_s$ as time slots. Here s represents the mission time. In practice, it is often desirable to find an assignment and a schedule that minimize the duration of the mission. A schedule maps tasks to *stop times*. That is, the stop time of a task $t \in \mathcal{T}$ is $TS(t)$. The stop time of a task is the time at which it finishes execution. The time at which task $t \in \mathcal{T}$ begins execution is its *start time* $TS(t) - D(t)$. The zero stop time is reserved and indicates that a task is not performed. That is, $TS(t) = 0$, where $t \in \mathcal{T}$ indicates that t is not

performed. This is physically significant as it indicates that regardless of the duration of the task, it must finish at time slot $0 \in \mathbb{T}_s$. Note that all tasks are assigned, but not necessarily performed. Here, we show that the OptDNSB Algorithm finds a schedule that minimizes the maximum over all task stop times. This is discussed in detail in Section 4.2.4. The following is a schedule for the example of Section 4.2,

$$\begin{aligned} TS_1 = \{&(t_1, 3), (t_2, 0), (t_3, 1), (t_4, 2), \\ &(t_5, 1), (t_6, 0), (t_7, 1), (t_8, 2)\} \end{aligned} \tag{4.62}$$

Scheduling constraints are formally defined, as in (3.10), as functions that map from the set of schedules to the set $\{false, true\}$. The number of constraints is N_s. If $p_m(TS) = true$ we say that the constraint is *satisfied* by the schedule, otherwise the constraint is *violated*. The constraints for the example of Section 4.2 are,

$$p_1 \equiv \sum_{i=2}^{4} I(TS(t_i) > 0) \geq 2, \tag{4.63}$$

$$\begin{aligned} p_{2-4} &\equiv TS(t_i) > 0 \Rightarrow TS(t_i) < TS(t_1) - D(t_1), \\ &i = 2, 3, 4, \end{aligned} \tag{4.64}$$

$$p_5 \equiv \sum_{i=6}^{8} I(TS(t_i) > 0) \geq 2, \tag{4.65}$$

$$\begin{aligned} p_{6-8} &\equiv TS(t_i) > 0 \Rightarrow TS(t_i) < TS(t_5) - D(t_5), \\ &i = 6, 7, 8, \end{aligned} \tag{4.66}$$

$$\begin{aligned} p_9 \equiv &[TS(t_1) < TS(t_5) - D(t_5)] \vee \\ &[TS(t_5) < TS(t_1) - D(t_1)], \end{aligned} \tag{4.67}$$

The function I in (4.63) and (4.65) is an indicator function that evaluates 1 if its argument is true and 0 if its argument is false. Constraints of (4.63) and (4.65) state that

at least two of the fires around each house should be extinguished. The constraints of (4.64) and (4.66) state that if a fire is extinguished, it must be extinguished before the people in the respective house are rescued.

The number of task clusters, as defined in (4.6), for the example of Section 4.2 is $N_s = 9$. The clusters representing the choice between the fires F1-F3 and the precedence constraints between F1-F3 and H1 are

$$\mathcal{T}_1 = \{t_2, t_3, t_4\}, \tag{4.68}$$

$$\mathcal{T}_2 = \{t_2, t_1\}, \tag{4.69}$$

$$\mathcal{T}_3 = \{t_3, t_1\}, \tag{4.70}$$

$$\mathcal{T}_4 = \{t_4, t_1\}; \tag{4.71}$$

the clusters representing the choice between the fires F4-F6 and the precedence constraints between F4-F6 and H2 are

$$\mathcal{T}_5 = \{t_6, t_7, t_8\}, \tag{4.72}$$

$$\mathcal{T}_6 = \{t_6, t_5\}, \tag{4.73}$$

$$\mathcal{T}_7 = \{t_7, t_5\}, \tag{4.74}$$

$$\mathcal{T}_8 = \{t_8, t_5\}; \tag{4.75}$$

and the cluster representing the one-task-at-a-time constraints between H1 and H2 is

$$\mathcal{T}_9 = \{t_1, t_5\}. \tag{4.76}$$

Figure 4.11 illustrates the cluster groups for the motivating example. Without loss of generality, we consider the scheduling of one cluster group.

Informally, if a task is not involved in any constraints that are violated it is said

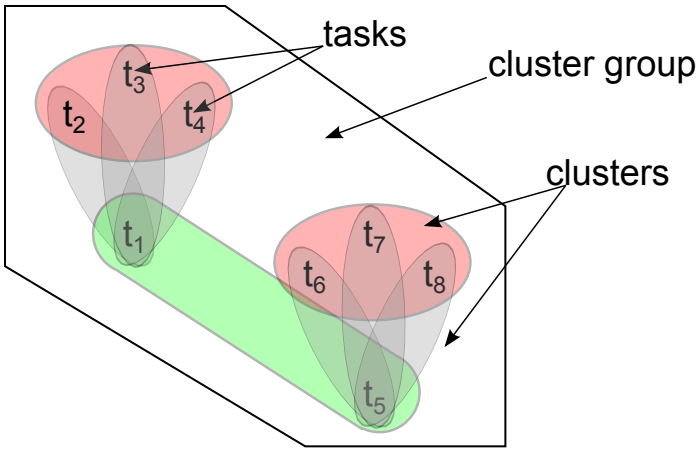


Figure 4.11: Clusters for the example scenario.

to be *consistent*. This idea is used to tell when all constraints a task is involved in have been satisfied. Formally, task $t_i \in \mathcal{T}$ is consistent if for every m where $t_i \in \mathcal{T}_m$, $p_m(TS) = true$, $m = 1, \ldots, N_s$. For the schedule in (4.62), $TS(t_5) = 1$ causes p_7 and p_8 in (4.66) to be violated. As such, tasks t_5, t_6, t_7, and t_8 are inconsistent.

Informally, a *feasible schedule* is a schedule that does not violate any constraints. Formally, feasible schedules are defined as follows.

Definition IV.7. Feasible schedule:
A schedule TS is called feasible if $\forall t_i \in \mathcal{T}$, t_i is consistent.

The schedule

$$\begin{aligned} TS_2 = \{&(t_1, 3), (t_2, 0), (t_3, 1), (t_4, 2), \\ &(t_5, 4), (t_6, 0), (t_7, 1), (t_8, 2)\} \end{aligned} \tag{4.77}$$

is feasible, while the schedule in (4.62) is not. The schedule in (4.77) is illustrated graphically using a Gantt chart in Figure 4.12.

In addition to feasibility, we desire a schedule that minimizes the time required to complete all tasks. Informally, a schedule that satisfies this is called *minimum-time*.

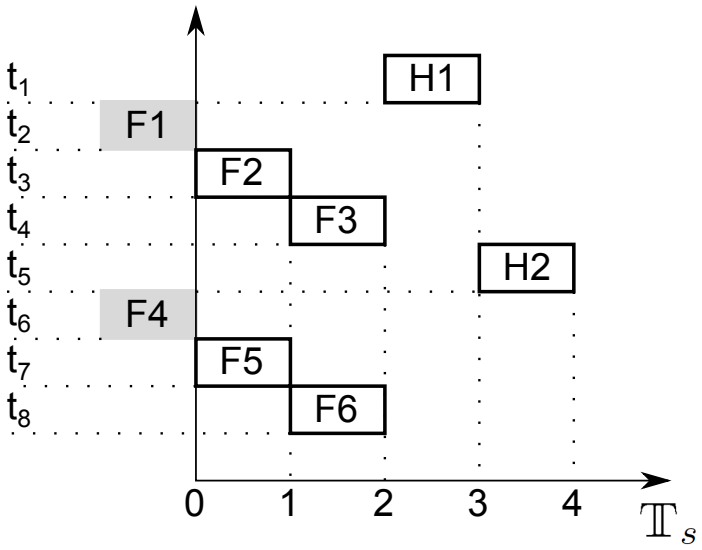


Figure 4.12: Gantt chart of the schedule in (4.77).

The objective function is

$$J(TS) = \max_{t_i \in \mathcal{T}} TS(t_i). \tag{4.78}$$

Formally, a minimum-time schedule $TS^* \in \mathbb{T}^{\mathcal{T}}_{max}$ is a solution to the optimization problem,

$$\min_{TS \in \mathbb{T}^{\mathcal{T}}_{max}} J(TS) \tag{4.79}$$

$$\text{s.t. } p_m(TS), m = 1, \ldots, N_s. \tag{4.80}$$

4.2.2 Problem Statement

Each agent $a_j \in \mathcal{A}$ is assumed to know the following data:

1. The set $TA^{-1}(a_j)$, i.e., the tasks that a_j is assigned to,
2. The set $\mathcal{N}_{a_j}$, i.e., the neighborhood of a_j on the communication graph $\mathcal{G}_c$,
3. The set $TA^{-1}(\mathcal{N}_{a_j})$, i.e., the tasks that the neighbors of a_j are assigned to,
4. For all $t \in TA^{-1}(a_j)$, p_m and $\mathcal{T}_m$ where $t \in \mathcal{T}_m$, i.e., all constraints and clusters that involve the tasks a_j is assigned to,

5. The function $D(t)$, $t \in TA^{-1}(a_j) \bigcup TA^{-1}(\mathcal{N}_{a_j})$, i.e., the duration function for tasks assigned to a_j and its neighbors,

6. The quantity s_{max}, i.e., the maximum allowable mission time,

where $m = 1, \ldots, N_{cl}$. Note that as TA is not necessarily injective, the inverse of TA provides a set of tasks rather than a single task.

The MADSP is for the agents in $\mathcal{A}$ to, given the data (1)-(6) and communication abilities defined by (4.12), find a TS^* that minimizes (4.79) subject to (4.80).

The assumed data (1)-(6) explicitly state the distributed nature of the MADSP. The knowledge of the set of tasks, the communication graph, the constraints, and the task durations is distributed. As a consequence, the agents must communicate to solve the MADSP. The difficulty of this scheduling problem is due to several factors: 1) the number of possible schedules is $\mathcal{O}((s_{max})^{N_t})$; 2) the scheduling constraints are general, predicate functions; 3) the objective function (4.78) is a nonlinear function of the $TS(t_i)$, that is, (4.78) is not a linear combination of the elements $TS(t_i)$, $t_i \in \mathcal{T}$; and 4) the problem data are distributed across a communication network of arbitrary topology.

4.2.3 Technical Approach

This section details the framework and the tools used to solve the task scheduling problem presented in Section 4.2.1. Similar to (4.19), for every multi-index (i, j) such that $(t_i, a_j) \in TA$, define a quadruple,

$$[t_i, a_j] = (States_{ij}, Start_{ij}, trans_{ij}, msgs_{ij}). \tag{4.81}$$

Here, $Start_{ij} \subset States_{ij}$ is the set of states at which process $[t_i, a_j]$ may begin operation;

$$trans_{ij} \quad : \quad \mathcal{M} \times States_{ij} \rightarrow States_{ij}, \tag{4.82}$$

$$msgs_{ij} \quad : \quad States_{ij} \rightarrow \mathcal{M}. \tag{4.83}$$

A process $[t_i, a_j]$ only sends messages to those processes $[t_k, a_l]$ where t_i and t_k share a cluster group. Let $Processes$ be the set of processes defined in (4.81).

Similar to (4.22), define the undirected process graph $\mathcal{G}_p = (Processes, \mathcal{E}_p)$, where

$$\mathcal{E}_p = \{\{[t_i, a_j], [t_k, a_l]\} \mid \{a_j, a_l\} \in \mathcal{E}_c\}. \tag{4.84}$$

The process $[t_i, a_j]$ is connected to the process $[t_k, a_l]$ if and only if agent a_j is connected to agent a_l by a communication link. The process graph for the motivating example is shown in Figure 4.13.

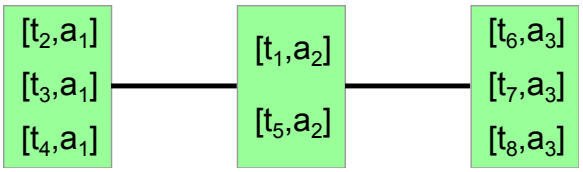


Figure 4.13: Process graph for the example of Section 4.2.

The vertex set of the process graph for the example of Section 4.2 is given by (4.58) and the edge set follows from (4.59) and (4.22).

The processes $[t_i, a_j] \in Processes$ form a distributed system in the sense specified in Section 4.2. This paper considers this distributed system under synchronous operation. That is, the processes each simultaneously update their state and then simultaneously send messages to their neighbors. Each iteration of computation and message transmission is referred to as a *round*.

When the problem data are not distributed among several agents a traditional

backtracking algorithm such as that presented in [95] can be used to find a feasible schedule. A basic variant of this algorithm is detailed in Appendix I. *Distributed backtracking* uses a set of processes to implement the traditional backtracking algorithm in a distributed way.

In a centralized setting, that is when the problem data are not distributed among several agents, a traditional backtracking algorithm such as that presented in [95] can be used to find a feasible schedule. *Distributed backtracking* uses a set of processes to implement the traditional backtracking algorithm in a distributed way. A basic variant of this algorithm is detailed in Algorithm 1.

In the OptDNSB Algorithm the elements $vs_{N_v}(i)$, $i = 1, \ldots, N_v$ in the sequence $vs_{N_v} \in \mathbb{D}^{N_v}$ represent scheduled stop times of tasks. Each element $vs_{N_v}(i)$, $i = 1, \ldots, N_v$ in the sequence $vs_{N_v} \in \mathbb{D}^{N_v}$ is associated with a process $[t_i, a_j]$, which is responsible for setting the value of $vs_{N_v}(i) \in \mathbb{D}$ and for running the *expand* and *backtrack* functions. The processes send appropriate messages to relay knowledge of the results of applying the *expand* and *backtrack* functions.

4.2.4 Solution Procedure

This section discusses the Optimal Distributed Non-Sequential Backtracking Algorithm used to solve the MADSP. The OptDNSB Algorithm also uses the specificity of the problem to find an optimal schedule. The OptDNSB Algorithm is augmented with the Minconflict Heuristic [81] that has been shown to improve backtracking efficiency.

Referring to Section 3.2, consider the sequence of variable values vs of length N_v, where for at least one of the p_m, $p_m(vs) = false$, $m = 1, \ldots, N_c$. Minconflict Backtracking iteratively improves vs, with respect to the number of violated constraints, by selecting a new value for a sequence element, $vs(j)$,$j = 1, \ldots, N_v$, that minimizes the number of constraint violations. Similarly to Algorithm 1, the selected elements

are removed from $untried_j \subseteq \mathcal{D}$. When $untried_j$ becomes empty, backtracking is performed. In this way, Minconflict Backtracking is able to achieve the efficiency of local search with the completeness of Backtracking.

The OptDNSB Algorithm exploits the benefits of Minconflict Backtracking [81] without appending additional constraints to the problem. This is in contrast to [120] where, in the worst case, the amount of memory required and the time required to evaluate the additional constraints can grow exponentially with the number of tasks. This is important because we deal with highly constrained problems where feasible solutions may not be numerous.

The OptDNSB Algorithm exploits parallelism and the specific properties of the scheduling problem in the following ways. Computation of the schedule times $TS(t)$ are computed in parallel by the processes $[t, TA(t)] \in Processes$. The backtracking algorithm operates using two basic message types, M_{ok} and M_{bt}, where M_{ok} messages relay the current values of the schedule and M_{bt} messages request that backtracking be done. In practice, these messages are sent by processes when a new result is computed. Constraint violations trigger backtracking, when backtracking is needed, it is requested (in parallel) and the violating portion of the solution space is pruned. The OptDNSB Algorithm minimizes mission time. Unlike other methods, it is not necessary to introduce additional constraints to be satisfied or functions to be minimized; this is accomplished by using a distributed extension of Minconflict Backtracking [81]. This minimization is done by iteratively searching over portions of the solution space where the mission time is a lower bound on the optimal mission time. The greatest known lower bound is communicated by processes to their neighbors in M_{ok} and M_{bt} messages. That is, only local information on the greatest lower bound is necessary. When a feasible schedule is found, this lower bound is tight and an optimal solution has been found. Hence, the OptDNSB Algorithm satisfies feasibility and optimality simultaneously.

Define a function called the priority function,

$$PR : \mathcal{T} \to \mathbb{N}, \tag{4.85}$$

that orders the set of tasks by assigning a natural number to them.

The $trans_{ij}$ and $msgs_{ij}$ Algorithms are run by processes $[t_i, a_j] \in Processes$ in synchronous rounds. The schedule at round r is referred to as TS_r. The state, $state_{ij}$, of process $[t_i, a_j]$ is as follows,

$$\begin{aligned} state_{ij} = (&j, i, highPrChange_{ij}, untried_{ij}, \\ &current_{ij}, violated_{ij}, consistent_{ij}, \\ &foundSolution_{ij}, maxHighPr_{ij}, minHighPr_{ij}, \\ &sendBt_{ij}, sendNS_{ij}, s_{ij}, \mathbb{T}_{ij}). \end{aligned} \tag{4.86}$$

At round r, for process $[t_i, a_j]$, the boolean quantity $highPrChange_{ij} = 1$ if $TS_r(t_k) \neq TS_{r-1}(t_k)$ for any $t_k \in \mathcal{C}_i$ where $PR(t_k) > PR(t_i)$. The vector $untried_{ij} \subseteq \mathbb{T}_{ij}$ is an ordered list of untried values in $\mathbb{T}_{ij}$. The integer quantity $current_{ij}$ is the index of the value in $untried_{ij}$ corresponding to $TS_r(t_i)$. The set $violated_{ij}$ is the set of those $t_k \in \mathcal{C}_i$ which are not consistent. If $violated_{ij} \subseteq \{t_k : PR(t_i) > PR(t_k)\}$, then $consistent_{ij} := 1$. The quantity $foundSolution_{ij} \in \{-1, 0, 1\}$. If $consistent_{ij} = 1$ and $consistent_{kl} = 1$ for all $[t_k, a_l]$ where $t_k \in \mathcal{C}_i$, $foundSolution_{ij} := 1$. If process $[t_i, t_j]$ decides that no solution exists, $foundSolution_{ij} := -1$; this is detailed in Algorithms 6 and 7. The tasks $maxHighPr_{ij}$ and $minHighPr_{ij}$, are defined as follows,

$$maxHighPr_{ij} = \arg\max_{t_k} PR(violated_{ij}), \tag{4.87}$$

$$\begin{aligned} minHighPr_{ij} = \arg\min_{t_k} \{&t_k \in violated_{ij} \\ &\text{s.t. } PR(t_k) > PR(t_i)\}. \end{aligned} \tag{4.88}$$

The boolean quantity $sendNS_{ij}$ tells whether s_{ij} has been incremented. The value s_{ij} is initialized as $s_{ij} = D(t_i)$.

The three types of messages are: $M_{ok} \in \mathcal{M}_{ok}$ which indicate that $consistent_{ij} = 1$, $M_{bt} \in \mathcal{M}_{bt}$ which request that backtracking be initiated, and $M_{noSol} \in \mathcal{M}_{noSol}$ which communicate that there does not exist a feasible schedule. The set of messages $\mathcal{M}$ is defined as follows,

$$\mathcal{M} = \mathcal{M}_{ok} \cup \mathcal{M}_{bt} \cup \mathcal{M}_{noSol}, \tag{4.89}$$

$$\begin{aligned} \mathcal{M}_{ok} = \{(j, i, TS_r(t_i), PR(t_i), \\ consistent_{ij}, TS_r(minHighPr_{ij})\}, \end{aligned} \tag{4.90}$$

$$\begin{aligned} \mathcal{M}_{bt} = \{(j, i, TS_r(t_i), PR(t_i), \\ consistent_{ij}, TS_r(minHighPr_{ij}))\}, \end{aligned} \tag{4.91}$$

$$\mathcal{M}_{noSol} = \{(s_{ij})\}, s_{ij} \in \mathbb{N}, \tag{4.92}$$

where i, j are such that $[t_i, a_j] \in Processes$.

The termination condition for this distributed algorithm applies to individual cluster groups. This results because the satisfiability of the constraints that correspond to one cluster group does not effect the satisfiability of the constraints that correspond to a different cluster group. Let the set $\mathcal{T}_{cg_1}$ be the set of tasks that belong to at least one pair that is a member of the cluster group CG_1. The termination of the OptDNSB Algorithm for those processes whose tasks share the cluster group CG_1 is:

Termination condition:
$\forall [t_i, a_j]$ s.t. $t_i \in \mathcal{T}_{cg_1}$, $foundSolution_{ij} = 1$ or $\forall [t_i, a_j]$ s.t. $t_i \in \mathcal{T}_{cg_1}$, $foundSolution_{ij} = -1$.

When one process sets $foundSolution_{ij} := -1$, it sends an M_{nosol} message to

its neighbors. Its neighbors then set $foundSolution_{ij} := -1$ and send it to their neighbors. This repeats until each process corresponding to a task in the cluster group has set $foundSolution_{ij} := -1$.

The function $findConsistent()$ is used by $trans_{ij}$ to set the value of $TS_r(t_i)$ to a value that does not violate any constraints. If there is no such value, it sets a value that minimizes the number of violated constraints as in [81]. The function $setBacktrack()$ saves the pair $(t_k, TS_r(t_k))$ so that a backtrack request may be sent by process $[t_i, a_j]$ to process $[t_k, a_l]$. The function $sendMsg()$ sends: an M_{ok} message to all neighbors whose respective tasks share a cluster group with t_i; an M_{bt} message to process $[minHighPr_{ij}, TA(minHighPr_{ij})]$; and an M_{noSol} message to all neighbors whose respective tasks share a cluster group with t_i.

For process $[t_i, a_j]$, Algorithm 6 is the $trans_{ij}$ function; it is responsible for finding consistent values of $TS(t_i)$ and recognizing that a feasible schedule has been found. Algorithm 7 performs the backtracking operation when it is called by $trans_{ij}$; it is responsible for incrementing s_{ij} and recognizing when there is no solution. Algorithm 8 is the $msgs_{ij}$ function; it is responsible for sending messages. The following description of the OptDNSB Algorithm refers to the classic Backtracking Algorithm (Algorithm 1) of Appendix I.

Line 11 of Algorithm 6 tests whether any values $TS_r(t_k)$, where $PR(t_k) > PR(t_i)$, have changed. If this is the case, $untried_{ij}$ is reset to an ordered list of the elements in $\mathbb{T}_{ij}$. Lines 5 and 6 of Algorithm 6 perform the operation of line 11 of Algorithm 1.

Lines 15-22 of Algorithm 6 tests whether task $PR(t_i)$ is less than all tasks in $violated_{ij}$. If this is the case, process $[t_i, a_j]$ is responsible for initiating backtracking by sending backtrack requests associated with the violated constraints it is involved in. In this way, either line 16 of Algorithm 6 finds a value of $TS_r(t_i)$ for which t_i is consistent or lines 18-20 of Algorithm 6 initiates backtracking. This performs the operation of line 7 of Algorithm 1.

Data: $state_{ij}$, M

if $M = M_{noSol}$ **then**
  if $M_{nosol}(s) > s_{ij}$ **then**
    $s_{ij} = M_{nosol}(s)$
    $\mathbb{T}_{ij} = \{0\} \cup \{D(t_i), \ldots, s_{ij}\}$
    $untried_{ij} = \mathbb{T}_{ij}$
    $sendNS_{ij} = 1$
  else if $M_{noSol}(s) = -1$ **then**
    $foundSolution := -1$
  end
end
if $highPrChange_{ij} = 1$ **then**
  $untried_{ij} := \mathbb{T}_{ij}$
  $consistent_{ij} := findConsistent()$
end
if $\min(PR(violated_{ij})) > PR(t_i)$ **then**
  $consistent_{ij} := findConsistent()$
  if $consistent_{ij} = 0$ **then**
    $setBacktrack(minHighPr_{ij},$
      $TS_r(minHighPr_{ij}))$
    $sendBt := 1$
  end
end
if $M = M_{bt}$ *and* $TS_r(t_i) = M(6)$ **then**
  $backTrack()$
end
if $\forall t_k \in \mathcal{C}_i, consistent_{kl} = 1$ *and* $consistent_{ij} = 1$ **then**
  $foundSolution_{ij} := 1$
end

Result: $state_{ij}$

Algorithm 6: $trans_{ij}$

```
1  if highPrChange_ij = 0 then
2      if | untried_ij |= 1 then
3          if PR(t_i) > PR(maxHighPr_ij) then
4              if s_ij = s_max then
5                  foundSolution_ij := −1
6              else
7                  sendNS = 1
8                  s_ij + +
9                  T_ij = {0} ∪ {D(t_i), ..., s_ij}
10                 untried_ij = T_ij
11             end
12         end
13         consistent_ij := findConsistent()
14         if consistent_ij = 0 then
15             setBacktrack(minHighPr_ij,
16                          TS_r(minHighPr_ij))
17             sendBt := 1
18         end
19     else
20         untried_ij := untried_ij \ untried_ij(current_ij)
21         consistent_ij := findConsistent()
22     end
23 end
```

Algorithm 7: $backTrack()$

```
   Data: state_ij
1  if foundSolution_ij = −1 or sendNS = 1 then
2      sendNS = 0
3      M = M_noSol
4  else
5      if sendBt = 1 then
6          sendBt = 0
7          M = M_bt
8          sendMsg(M)
9      end
10     M = M_ok
11 end
12 sendMsg(M)
   Result: M
```

Algorithm 8: $msgs_{ij}$

Lines 23-25 of Algorithm 6 respond to requests for backtracking. The element $M(6)$ refers to the finish time that must be changed. Note that backtracking only proceeds if $TS_r(t_i)$ is still equal $M(6)$. If all of the possible values for $TS_r(t_i) \in \mathbb{T}_{ij}$ have been exhausted, a backtracking request is sent to process $[minHighPr_{ij}, TA(minHighPr_{ij})]$, lines 16-18 of Algorithm 7. If $untried_{ij}$ contains a feasible value, the current value $TS_r(t_i)$ is removed from $untried_{ij}$ and a new value is found using $findConsistent()$ by lines 21 and 22 of Algorithm 7. This prunes portions of the search space that do not contain a feasible schedule, i.e., performs the operation of line 10 of Algorithm 1.

Lines 26-28 of Algorithm 6 test whether $consistent_{kl} = 0$ for any $[t_k, a_l] \in \mathcal{N}_{[t_i,a_j]}$ where there exists a $\mathcal{T}_m$ s.t. $t_i, t_k \in \mathcal{T}_m$. If $consistent_{kl} = 1$ for all such $[t_k, a_l]$, process $[t_i, a_j]$ declares that it has found a solution.

Line 4 of Algorithm 7 tests whether $PR(t_i) > PR(t_k)$ for all $t_k \in violated_{ij}$. If this is the case and process $[t_i, a_j]$ has exhausted all possible start times in $\mathbb{T}_{ij}$, process $[t_i, a_j]$ declares that there is no solution for the current value of s_{ij}. If $s_{ij} = s_{max}$ then there is no solution for the specified maximum schedule range.

4.2.5 Analysis of Optimal Distributed Non-Sequential Backtracking

This subsection analyzes the Optimal Distributed Non-Sequential Backtracking Algorithm. The OptDNSB Algorithm is proven correct, complete, and optimal. The algorithm is correct in that it outputs that a schedule is feasible if and only if the schedule is feasible. The algorithm is complete in that it either returns a feasible schedule or decides that there is no such schedule exists in finite time. The algorithm is optimal in that if a feasible schedule exists it returns a minimum-time feasible schedule. Lemmas IV.8-IV.11 state the algorithm's correctness, Lemma IV.12 and Theorem IV.13 states the algorithm's completeness and optimality. All lemmas apply to processes whose respective tasks belong to a single cluster group.

4.2.6 Correctness

Correctness of the OptDNSB Algorithm refers to the requirement that the algorithm output that a schedule is feasible if and only if a schedule is feasible, or output that no such schedule exists if and only if there exists no feasible schedule. Lemma IV.8 states the first part of the correctness of the OptDNSB Algorithm as the equivalence of $consistent_{ij} = 1$ for all processes to a feasible schedule being found.

Lemma IV.8.

$\forall [t_i, a_j] \in Processes$, $consistent_{ij} = 1$ *if and only if* TS *feasible.*

Proof. Lemma IV.8

"$\rightarrow$" Assume that for all $[t_i, a_j] \in Processes$, $consistent_{ij} = 1$. This implies that $p_m(TS) = 1, m = 1, \ldots, N_{cl}$, which implies that TS is feasible.

"$\leftarrow$" Assume TS is feasible. A feasible TS implies that $\forall [t_i, a_j] \in Processes$, $violated_{ij} = \emptyset$. This implies that $violated_{ij} \setminus \{t_k : PR(t_i) > PR(t_k)\} = \emptyset$, which implies that for all $[t_i, a_j] \in Processes$, $consistent_{ij} = 1$. □

The second part of the correctness of the OptDNSB Algorithm is its ability to correctly disregard portions of the solution space that contain no feasible schedules. Lemma IV.9 states that requests for backtracking are only sent when there exists a subset of TS that cannot be part of a feasible schedule.

Lemma IV.9.

If a process $[t_i, a_j] \in Processes$ *sends an* $\mathcal{M}_{bt}$ *message at round* r*, then there exists a* $p_m(TS_{r-1}) = false$ *where* $t_i \in \mathcal{T}_m$ *and* $|untried_{ij}| = 1$*. And if a process* $[t_k, a_l] \in Processes$ *backtracks in response to an* $\mathcal{M}_{bt}$ *message, then there exists a* p_m*:*

$p_m(TS_r) = false.$

Proof. Lemma IV.9 1) This proof is by contradiction. Let TS_r be a feasible schedule and $PR(t_i) < PR(t_k)$. Assume that an M_{bt} message has been sent by process $[t_i, TA(t_i)]$ to $[t_k, TA(t_k)]$. By lines 3, 14, and 15 of Algorithm 7, the only remaining element in $untried_{ij}$ results in t_i being inconsistent. That is, there exists a constraint p_1 where $t_i \in \mathcal{T}_1$ and $p_1(TS_r) = false$. This is a contradiction, TS_r cannot be feasible and violate a constraint.
2) By line 23 in Algorithm 6, $backTrack()$ only executes if $TS_r(t_i) = M(6)$. By (1), $TS_r(t_i) = M(6)$ implies that $\exists p_1$: $p_1(TS_r) = false$. Hence, Lemma IV.9. □

This lemma is important because it ensures that the OptDNSB Algorithm never prunes a portion of the search space that may contain a feasible schedule. Here, if the set $untried_{ij}$ has cardinality equal 1 and $consistent_{ij} = 0$ for the corresponding value of $TS(t_i)$, then process $[t_i, a_j]$ has exhausted all values in $\mathbb{T}_{ij}$ and backtracking must be initiated. Lemma IV.10 states that the OptDNSB Algorithm expands the interval $\mathbb{T}_s \subseteq \{0, \ldots, s\}$ only when no solution exists for the current value of $s \leq s_{max}$.

Lemma IV.10.

A process $[t_i, a_j] \in Processes$ increments s_{ij} if and only if there does not exist a feasible schedule where $\forall t \in \mathcal{T}$, $TS(t) \leq s_{ij}$.

Proof. Lemma IV.10

"$\rightarrow$" Assume some process $[t_i, a_j] \in Processes$ increments s_{ij}. This implies that all values $TS(t_i) \in \mathbb{T}_{ij}$, by lines 3 and 7 in Algorithm 7 and Lemma IV.9, have been correctly eliminated as possibly belonging to a feasible schedule.

"←" Assume there does not exist a feasible schedule where $s = s_{ij}$. That is, there exists at least one constraint $p_1 : \forall TS \in \mathbb{T}_s^{\mathcal{T}}$, $p_1(TS) = 0$. Without loss of generality we can consider a single unsatisfiable constraint; this is because a set of constraints that is unsatisfiable can be replaced by a single unsatisfiable constraint for analysis purposes. Let $t_i = \arg\max_{t \in \mathcal{T}_1} PR(\mathcal{T}_1)$, eventually $[t_i, a_j]$ will receive M_{bt} messages for all values in $\mathbb{T}_s$. This implies that eventually, $[t_i, a_j]$ will set $s_{ij} = s_{ij} + 1$. □

Lemma IV.11 extends Lemma IV.10 to guarantee that the OptDNSB Algorithm outputs that no solution exists for $\mathbb{T}_s = \mathbb{T}_{max} \subseteq \{0, \ldots, s_{max}\}$ only when no solution exists for the prespecified maximum value of s (i.e., $s = s_{max}$).

Lemma IV.11.

A process $[t_i, a_j] \in Processes$ sets $foundSolution_{ij} = -1$ if and only if there does not exist a feasible schedule.

Proof. Lemma IV.11

Consider process $[t_i, a_j] \in Processes$ where $s_{ij} = s_{max}$. Let p_1 be unsatisfiable and $t_i = \arg\max_{t \in \mathcal{T}_1} PR(\mathcal{T}_1)$. By Lemma IV.10, $[t_i, a_j]$ will eventually set $s_{ij} = s_{ij} + 1$. Lines 5-7 of Algorithm 7 will set $foundSolution = -1$. □

4.2.7 Completeness and Optimality

Completeness of the OptDNSB Algorithm refers to the algorithm's ability to terminate with a solution to the constrained optimization problem of (4.79) and (4.80) or decide that no solution exists in finite time. The completeness of the OptDNSB Algorithm is formally given by Lemma IV.12 and Theorem IV.13. Lemma IV.12 states that each process within a cluster group will eventually output that a solution has been found or that no feasible schedule exists. Lemma IV.12 also states that each

such process will output the same answer.

Lemma IV.12.

All processes $[t_i, a_j] \in Processes$ *in a cluster group eventually either: 1) all set* $foundSolution_{ij} = 1$*, or 2) all set* $foundSolution_{ij} = -1$*.*

Proof. Lemma IV.12

1) By Lemmas IV.9 and IV.8, OptDNSB will never incorrectly skip a feasible schedule and OptDNSB will always correctly identify a feasible schedule. Hence, if OptDNSB terminates, it will terminate with a feasible schedule or correctly output that no feasible schedule exists.

2) Line 21 in Algorithm 7 prunes infeasible schedules from the search space. By Lemma IV.10 and lines 2-3 in Algorithm 6 and line 9 in Algorithm 7, s_{ij} increases monotonically for all $[t_i, a_j] \in Processes$. Hence, the OptDNSB Algorithm progresses monotonically through the set of schedules $\mathbb{T}^{\mathcal{T}}_{max}$. The set of schedules $\mathbb{T}^{\mathcal{T}}_{max}$ is finite. Hence, OptDNSB will eventually terminate.

By 1) and 2), OptDNSB will terminate and will terminate with a feasible schedule, i.e., $foundSolution_{ij} = 1$ at $s = s_{max}$, or correctly output that no feasible schedule exists, i.e., $foundSolution = -1$ at $s = s_{max}$, $[t_i, a_j] \in Processes$. □

Theorem IV.13 states the completeness and optimality of the OptDNSB Algorithm.

Theorem IV.13.

The OptDNSB Algorithm terminates in finite time with a minimum-time schedule.

Proof. Theorem IV.13

The quantity s_{ij} is initialized as $D(t_i)$ where the $D(t_i) \leq \max_i D(t_i)$, $t_i \in \mathcal{T}$ are all lower bounds on the cost function of (4.78). By Lemma IV.10 and lines 2-3 in Algorithm 6 and line 9 in Algorithm 7, s_{ij} increases monotonically for all $[t_i, a_j] \in Processes$.

Lemma IV.12 shows the completeness of OptDNSB. We have the following: by Lemma IV.10, s_{ij} is incremented when no solution exists for $\mathbb{T}_{ij} = \{0\} \bigcup \{D(t_i), \ldots, s_{ij}\}$; lines 2-3 in Algorithm 6 and lines 1 and 3 of Algorithm 8 ensure that all processes will increment s_{ij}; by completeness, s_{ij} will be incremented until an s is found such that a feasible schedule exists. That is, the first feasible schedule found is an optimal schedule.

Hence, OptDNSB will terminate when and only when the lower bound s is tight, i.e., when an optimal schedule has been found. □

By Lemmas IV.8-IV.12 and Theorem IV.13, the OptDNSB Algorithm is correct, complete, and finds a minimum-time schedule. The OptDNSB Algorithm does this while reducing computational complexity when possible. The mechanism for this is in the fact that the co-domain of the schedule function TS is $\mathbb{T}_s$ which represents the finish times of the tasks in $\mathcal{T}$. In traditional distributed backtracking searches, the size of this co-domain is fixed in advance [121]. In this scheduling application, the parameter s is increased incrementally such that it remains a lower bound on the minimum-time. In essence, the OptDNSB Algorithm searches for a feasible schedule over sets of monotonically increasing schedule lengths until a feasible schedule is found.

4.2.8 Simulation Examples

Here, we demonstrate the solution of the example problem in Section 4.2. Consider the duration function (4.61) and the constraints (4.63)-(4.67). The optimal schedule

found by the OptDNSB Algorithm is that of (4.77). The algorithm finds this schedule in 25 rounds. The optimal objective function value is $J(TS_2) = 4$. Note that several feasible schedules achieve the same value of the objective function in (4.78). For this example, the schedules

$$\begin{aligned} TS_3 = \{&(t_1, 3), (t_2, 2), (t_3, 1), (t_4, 0), \\ &(t_5, 4), (t_6, 2), (t_7, 1), (t_8, 0)\} \end{aligned} \tag{4.93}$$

and

$$\begin{aligned} TS_4 = \{&(t_1, 3), (t_2, 1), (t_3, 0), (t_4, 2), \\ &(t_5, 4), (t_6, 1), (t_7, 0), (t_8, 2)\} \end{aligned} \tag{4.94}$$

are both feasible and optimal with $J(TS_3) = J(TS_4) = 4$. Hence, optimal schedules are generally not unique. In addition, aircraft EV1 and EV2 are scheduled to simultaneously extinguish fires F2 and F5 followed by F3 and F6, R1 is scheduled to rescue H1 and H2 immediately following. It is expected that EV1 and EV2 would be scheduled to act simultaneously to reduce the overall time needed. Vehicle travel time is not modeled here.

Consider the follow duration function,

$$\begin{aligned} D = \{&(t_1, 1), (t_2, 1), (t_3, 1), (t_4, 1), \\ &(t_5, 1), (t_6, 2), (t_7, 2), (t_8, 2)\}, \end{aligned} \tag{4.95}$$

where fires F4-F6 will take twice as long to extinguish as F1-F3. The optimal schedule found by the OptDNSB Algorithm with the duration function of (4.95) is

$$\begin{aligned} TS_5 = \{&(t_1, 4), (t_2, 2), (t_3, 1), (t_4, 0), \\ &(t_5, 5), (t_6, 0), (t_7, 2), (t_8, 4)\}. \end{aligned} \tag{4.96}$$

The schedule in (4.96) is illustrated graphically using the Gantt chart in Figure 4.14. The schedule in (4.96) is found in 22 rounds. For this example, $J(TS_5) = 5$, which

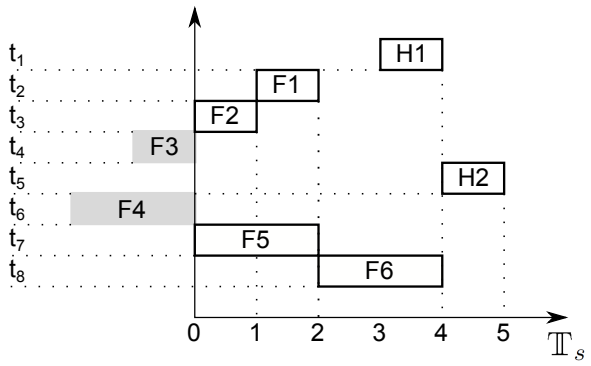


Figure 4.14: Gantt chart of the schedule in (4.96).

results from the extra time needed to extinguish fires F5 and F6. Note that R1 is scheduled to finish the rescue of H1 at $TS(t_1) = 4$. It is expected that the rescue of H1 would be scheduled to precede H2 due to the extra time needed to extinguish F5 and F6. However, one effect of using the objective function in (4.78) is that H1 is not scheduled to be rescued as quickly as possible because this would not effect the value of the objective function. That is, for

$$\begin{aligned} TS_6 = \{&(t_1, 3), (t_2, 2), (t_3, 1), (t_4, 0), \\ &(t_5, 5), (t_6, 0), (t_7, 2), (t_8, 4)\}, \end{aligned} \tag{4.97}$$

the objective function $J(TS_6) = J(TS_5) = 5$.

4.2.9 Big-O Complexity Analysis

The complexity of the OptDNSB Algorithm is as follows. The OptDNSB Algorithm iteratively solves a constraint satisfaction problem. In the worst case the algorithm incurs a complexity of $\mathcal{O}((s^*)^{N_t})$ each iteration. Here, $s^* \leq s_{max}$ is the minimum of (4.78). The resulting worst-case computational complexity of the OptDNSB Algorithm is $\mathcal{O}((s^*)^{N_t+1})$. In contrast, if $\mathbb{T}_s$ were fixed at $\mathbb{T}_s = \mathbb{T}_{max}$, the worst-case

complexity would be $\mathcal{O}(s_{max}^{N_t})$.

Now we can see that if $s^* = s_{max}$, the OptDNSB Algorithm incurs a higher worst-case complexity. However, in practice s^* is unknown and while an upper bound may be available, it may not be tight. For such instances, the OptDNSB Algorithm can provide minimum-time solutions with lower computational complexity.

When tasks belong to different cluster groups, they are not related by constraints. The function $msgs_{ij}$ only sends messages to those processes $[t_k, a_l] \in \mathcal{N}_{[t_i,a_j]}$ where t_i and t_k share a cluster group. Hence, processes in one cluster group cannot affect the values of $consistent_{ij}$, s_{ij}, and $foundSolution_{ij}$ of a process in a different cluster group. Each cluster group therefore represents a distinct *scheduling subproblem*. The independence of scheduling subproblems decreases the computational complexity of solving the MADSP. Let c be the maximum number of tasks in any cluster group. Additionally, let each process execute in parallel, i.e., a round executes in $\mathcal{O}(1)$ time. Then the worst-case complexity of the OptDNSB Algorithm is $\mathcal{O}((s^*)^{c+1})$. For a given maximum cluster group size, the worst case complexity is independent of N_t. This is consistent with the results of [95].

4.2.10 Experimental Complexity Analysis

This section presents experimental results on the complexity of the OptDNSB Algorithm. Figure 4.15 shows the mean and standard deviation of the number of rounds needed to solve random instances of the MADSP as a function of the number of tasks. The following discussion presents the methodology of constructing these problem instances.

Without loss of generality, we consider problem instances with a single cluster group. The development of the OptDNSB Algorithm considers constraints of any order, i.e., order $\mid \mathcal{T}_m \mid < \infty$, $m = 1, \ldots, N_{cl}$. For instance, a constraint to choose M from N tasks, $M < N$, is of order N. A common type of scheduling constraint is the

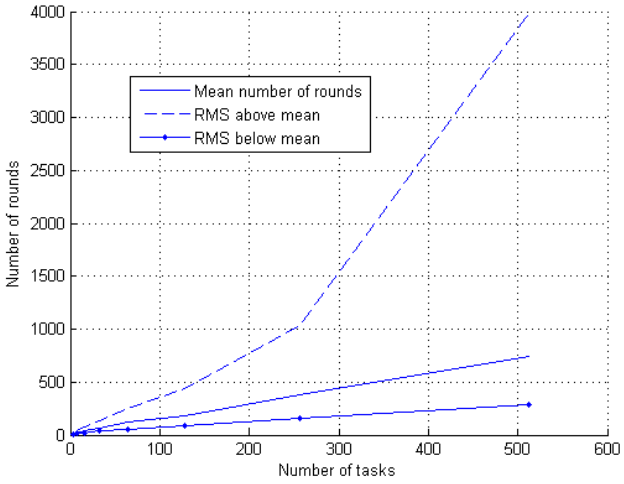


Figure 4.15: Number of rounds needed to find an optimal schedule. Data points represent mean and standard deviation of the number of rounds for 50 randomly generated scheduling problems.

precedence constraint,

$$p_m \equiv (TS(t_i) \leq TS(t_j) - D(t_j)), m = 1, \ldots, N_{cl}, \tag{4.98}$$

where $t_i \in \mathcal{T}$, $t_j \in \mathcal{T}$, and p_m is satisfied if and only if t_i finishes before (or when) t_j begins. Note that precedence constraints are of order two, i.e., $\mid \mathcal{T}_m \mid = 2$. Consider a graph

$$\mathcal{G}_t = (\mathcal{T}, \mathcal{E}_t), \tag{4.99}$$

where the edges in $\mathcal{E}_t$ correspond to precedence constraints. Considering a single cluster group restricts this graph to be connected. We restrict $\mathcal{G}_t$ to be a spanning tree so that no sequence of precedence constraints results in a contradiction. A problem instance is constructed by first constructing a random spanning tree on the node-set $\mathcal{T}$. The edges of the spanning tree correspond to the constraints of the problem instance. Spanning trees are constructed uniformly from the set of possible spanning trees. This technique produces random precedence-constrained scheduling problem instances.

Each problem instance is solved to optimality using the OptDNSB Algorithm. Each data point in Figure 4.15 represents the sample-mean and standard deviation of 50 computational experiments. The slope of the line representing the mean number of rounds is approximately constant. This indicates a near-linear average-case complexity with respect to the number of rounds for this class of problem.

With regards to communication complexity, the $sendMsg()$ function in $msgs_{ij}$ (Algorithm 8) sends, at most, two messages to $\mid \mathcal{C}_i \mid$ of the processes in $\mathcal{N}_{[t_i,a_j]}$ at each round where $[t_i, a_j] \in Processes$. Hence, each process sends, at most, $2\cdot \mid \mathcal{C}_i \mid$ messages at each round. Let the number of rounds needed to find a solution to the MADSP be given by the function $f_r(N_t, s^*)$. It follows that the number of messages sent, the communication complexity, is $\mathcal{O}\left(2\cdot \mid \mathcal{C}_i \mid \cdot f_r(N_t, s^*)\right)$. For the big-O analysis this gives an exponential communication complexity. For the average-case analysis in Figure 4.15 the communication complexity is near-linear.

4.2.11 The Coupling Between Assignments and Schedules

The motivation for task assignment and task scheduling decoupling is related to the ability to solve the distributed task assignment and task scheduling problems when the communication network has an arbitrary topology. This section details the properties that a distributed assignment algorithm must satisfy so that a distributed scheduling algorithm can run concurrently.

The capability relation for the example of Section 4.2 is

$$\begin{aligned} Capability = \{&(t_1, a_2), (t_2, a_1), (t_2, a_3), (t_3, a_1), (t_3, a_3), \\ &(t_4, a_1), (t_4, a_3), (t_5, a_2), (t_6, a_1), (t_6, a_3), \\ &(t_7, a_1), (t_7, a_3), (t_8, a_1), (t_8, a_3)\}. \end{aligned} \tag{4.100}$$

Note that (4.100) states that: EV1 and EV2 are capable of performing the fire extinguishing tasks t_2, t_3, t_4, t_6, t_7, and t_8; and R1 is capable of performing the rescue

tasks t_1 and t_5. The task assignment of (4.3) is feasible with respect to capability and clustering, i.e., it is a feasible task assignment.

Rather than have processes wait for a feasible assignment to begin scheduling, we allow processes to begin scheduling when they become schedulable. Informally, a constraint p_m is schedulable if the clustering constraint associated with $\mathcal{T}_m$ is satisfied. Formally, constraint schedulability is defined as follows.

Definition IV.14. Schedulable Constraint:
Constraint p_m is schedulable if $(\mathcal{A}, \mathcal{E}_c)|_{TA(\mathcal{T}_m)}$.

A process $[t_i, a_j]$ is schedulable with respect to the constraint p_m, where $t_i \in \mathcal{T}_m$, if its neighbors are assigned all other tasks $t_k \in \mathcal{T}_m$. A process $[t_i, a_j] \in Processes$ is schedulable with respect to a constraint $\mathcal{T}_m$ if it is in communication with all processes whose agents are assigned the tasks it shares cluster $\mathcal{T}_m$ with.

Definition IV.15. Schedulable Process:
A process $[t_i, a_j]$ is schedulable with respect to p_m if $(t_i, a_j) \in TA$ and $\forall t_k \in \mathcal{T}_m, k \neq i$, $\exists [t_k, a_l] \in \mathcal{N}_{ij}$ s.t. $(t_k, a_l) \in TA$.

The ability of processes to determine schedulability locally is discussed as follows. Define a function

$$SC : Processes \times \{1, \ldots, N_s\} \rightarrow \{false, true\}, \tag{4.101}$$

to be an indicator function that evaluates *true* if the input process is schedulable with respect to constraint $p_m, m \in \{1, \ldots, N_s\}$ and $false$ if not. Processes use this

function to determine schedulability locally. Lemma IV.16 says that schedulability of a constraint p_m can be determined locally by a process $[t_i, a_j]$ where $t_i \in \mathcal{T}_m$ if process $[t_i, a_j]$ has access to SC.

Lemma IV.16. *Local Schedulability*

$\forall t_i \in \mathcal{T}_m$ $SC([t_i, a_j], m) = true$ *if and only if* p_m *is schedulable.*

Proof. Lemma IV.16

"$\rightarrow$" Assume p_m is schedulable, that is $\{TA(t_i), TA(t_k)\} \in \mathcal{E}_c$ for all $t_i, t_k \in \mathcal{T}_m$. This implies that $\{[t_i, TA(t_i)], [t_k, TA(t_k)]\} \in \mathcal{E}_p$ for all $t_i, t_k \in \mathcal{T}_m$. That is, $[t_i, TA(t_i)]$ and $[t_k, TA(t_k)]$ are schedulable for all $t_i, t_k \in \mathcal{T}_m$ which implies that $SC([t_i, a_j], m) = 1$ $\forall t_i \in \mathcal{T}_m$.

"$\leftarrow$" Assume $SC([t_i, a_j], m) = 1$ $\forall t_i \in \mathcal{T}_m$. That is, for all $t_i, t_k \in \mathcal{T}_m$, $[t_i, TA(t_i)]$ and $[t_k, TA(t_k)]$ are schedulable. This implies that $\{[t_i, TA(t_i)], [t_k, TA(t_k)]\} \in \mathcal{E}_p$ for all $t_i, t_k \in \mathcal{T}_m$ and that $\{TA(t_i), TA(t_k)\} \in \mathcal{E}_c$ for all $t_i, t_k \in \mathcal{T}_m$. This implies that p_m is schedulable. □

It is important that processes determine schedulability locally before scheduling tasks. Satisfying schedulability for a constraint p_m is equivalent to satisfying the constraints of (4.14) for $\mathcal{T}_m$. Once schedulability is satisfied the OptDNSB Algorithm can be used to find a feasible schedule. This takes advantage of the separation of assignment and scheduling in cases where scheduling constraints do not involve the task assignment. It is important to show that scheduling tasks as constraints become schedulable does not adversely effect the completeness of the OptDNSB Algorithm.

Without loss of generality, let the constraints $p_1, \ldots, p_{N_{cl}}$ become schedulable in numerical order. That is, p_1 becomes schedulable, then p_2, etc. Let the sets

$$\mathbb{T}^{\mathcal{T}}_{s_m} \subseteq \mathbb{T}^{\mathcal{T}}_{s}, m = 1, \ldots, N_{cl}, \tag{4.102}$$

be such that $\mathbb{T}^{\mathcal{T}}_{s_m}$ is the set of schedules that are feasible with respect to constraints $p_1, \ldots, p_m$. That is: $\forall TS \in \mathbb{T}^{\mathcal{T}}_{s_1}$, $p_1(TS) = true$; $\forall TS \in \mathbb{T}^{\mathcal{T}}_{s_2}$, $p_1(TS) \wedge p_2(TS) = true$; ... and $\forall TS \in \mathbb{T}^{\mathcal{T}}_{s_m}$, $p_1(TS) \wedge \ldots \wedge p_m(TS) = true$. The sets in (4.102) are nested in the following way,

$$\mathbb{T}^{\mathcal{T}}_{s_m} \subseteq \mathbb{T}^{\mathcal{T}}_{s_{m-1}} \subseteq \ldots \subseteq \mathbb{T}^{\mathcal{T}}_{s_1}, m = 1, \ldots, N_{cl}. \tag{4.103}$$

Note that increasing the number of constraints can never make the problem less constrained and that increasing the number of constraints cannot reduce the value of the optimal cost in (4.79). Lemma IV.17 says that the number of constraints can be increased and completeness is maintained.

Lemma IV.17. *Addition of constraints*

The OptDNSB Algorithm remains correct when solving the problem of (4.79) and (4.80) over the sets $\mathbb{T}^{\mathcal{T}}_{s_1}, \ldots, \mathbb{T}^{\mathcal{T}}_{s_m}$ in sequence.

Proof. Lemma IV.17

From Theorem IV.13, OptDNSB is correct, complete, and optimal, i.e., it solves (4.79), (4.80). This result is independent of the number of constraints in (4.80). Let there exist a feasible schedule $TS^* \in \mathbb{T}^{\mathcal{T}}_{max}$.

This proof is by induction. Consider $e = k + 1$ executions of OptDNSB where execution e+1 is initialized with the last schedule of the previous execution $TS = TS_e$. Let $m = 1$ in (4.80) for execution $e = 1$. By completeness of OptDNSB, no feasible schedule in $\mathbb{T}^{\mathcal{T}}_{s_1}$ has been missed.

Consider execution $e = k$ and $m = 1, \ldots, k$ in (4.80). Assume $TS^* \in \mathbb{T}^{\mathcal{T}}_{s_k}$ has been found; by completeness and optimality, no feasible schedule has been skipped. Consider execution $e = k + 1$ and $m = 1, \ldots, k + 1$ in (4.80) with the initial schedule

$TS_e = TS^* \in \mathbb{T}^{\mathcal{T}}_{s_k}$. By completeness and optimality, OptDNSB finds $TS^* \in \mathbb{T}^{\mathcal{T}}_{s_{k+1}}$.

By induction, at execution $e = N_{cl}$, OptDNSB will find $TS^* \in \mathbb{T}^{\mathcal{T}}_{s_{N_{cl}}}$. If no solution exists in $\mathbb{T}^{\mathcal{T}}_{s_k}$ where $k < N_{cl}$, there exists no solution in $\mathbb{T}^{\mathcal{T}}_{s_{N_{cl}}}$. By completeness, OptDNSB will output that there is no solution. □

Lemma IV.17 does not address the removal of constraints. Constraint removal must be addressed because we only require that the assignment algorithm eventually find a feasible assignment. Theorem IV.18 completes this point. Recall that we only consider static problem instances. That is, the tasks, agents, communication graph, duration function, the capability relation, and the constraint functions do not change.

Theorem IV.18. *Removal of constraints*

If there exists a round r' where $\forall r > r'$ and $\forall p_m$, p_m is schedulable, then the schedulability of constraints may change and the OptDNSB Algorithm remains complete.

Proof. Theorem IV.18

Removing constraints relaxes the problem instance, i.e., removing constraint p_m results in searching for a schedule $TS \in \mathbb{T}^{\mathcal{T}}_{s_{m-1}}$ where $\mathbb{T}^{\mathcal{T}}_{s_m} \subseteq \mathbb{T}^{\mathcal{T}}_{s_{m-1}}$. The eventual schedulability of all constraints implies that any schedules that are pruned before the removal of constraint p_m would be pruned when p_m is eventually returned to schedulability. Additionally, by the completeness of OptDNSB, no solution in $\mathbb{T}^{\mathcal{T}}_{s_{m-1}}$ will be missed while p_m remains unschedulable. Hence, by Lemma IV.17 and the fact that infeasible schedules resulting from p_m will be pruned, constraints may be removed from and returned to schedulability without affecting the completeness of OptDNSB. □

The importance of Theorem IV.18 is that if a distributed assignment algorithm finds a task assignment that is feasible with respect to capability and clustering,

by Lemma IV.17 and Theorem IV.18, the distributed scheduling algorithm remains complete.

The results of this chapter demonstrate that task assignment and minimum-time task scheduling can be accomplished by distributed agents using a network with arbitrary topology. The strategy used here first satisfies communication constraints, then guarantees an optimal and feasible task schedule. This strategy allows expressiveness in the problem description to be sacrificed to reduce the communication link requirements of solving the task assignment and task scheduling problem. This strategy is useful when the task assignment constraints and the task scheduling constraints are independent. This strategy works because adding and removing scheduling constraints, that must be satisfied eventually, to and from the task scheduling problem cannot cause the task scheduling algorithm to erroneously prune feasible portions of the search space.

Several quantities influence the difficulty of these problems. The clustering index is the quantity that indicates the connectedness required to satisfy the clustering constraints. The difficulty of the CDAP increases with the value of the clustering index. Increasing the number of agents increases the complexity of the SBA and increasing the number of tasks increases the complexity of both the SBA and the OptDNSB Algorithm. Both the SBA and the OptDNSB Algorithm exhibit reasonable complexity. This is justified by the linear and less than linear complexity exhibited by these algorithms when solving randomized problem instances.

CHAPTER V

Summary of Accomplishments and Future Work

The contributions of this dissertation are in the areas of centralized and distributed task assignment and task scheduling. The centralized task assignment and task scheduling problem solved in Chapter III is a vehicle routing problem of interest in military mission planning. The formulation is able to incorporate common mission constraints and express an important objective function, mission time. The polynomial-time algorithm developed for solving the problem is the Tabu/2-opt heuristic. This repair heuristic exploits a separation of the task assignment and task schedule to quickly (greedily) improve the quality of an initial candidate solution. The solution quality was compared to a Branch and Bound solution technique that provides optimal solutions for small problem instances. This analysis showed that while the combined heuristic is suboptimal, the resulting solutions average 23% deviation from optimal. The final solutions for a typical instance with $\mathcal{O}(10)$ tasks are computed in $\mathcal{O}(10)$ seconds on a modest computer, but the initial feasible solution is generated in $\mathcal{O}(0.001)$ seconds. This time requirement is quite acceptable for real-time operations where the mission execution time can be an hour or more.

A new measurement of solution quality for combinatorial problems was presented. This method considers task assignment and task scheduling problems, in particular, those that are NP-hard. This technique exploits the fact that feasible candidate

solutions to many task assignment and task scheduling problems can be computed quickly. The method also exploits the fact that if the costs of those solutions (when sampled independently) follow a distribution with a single mode, the distribution can be transformed into a Gaussian distribution. This allows for quantitative comparison between candidate solutions and the space of possible solutions. This is useful because it allows us to quantitatively measure the quality of a solution to a problem for which no analytical guarantees exist. The quality measurement is in the form of the probability of finding a better solution. This technique gives the solution quality relative to the space of possible solutions.

It contributes specifically to the literature of distributed task assignment and task scheduling where the communication network may have an incomplete topology that is only locally known. This dissertation developed the Communication-Constrained Distributed Assignment Problem. This problem was converted to a minimization problem and the Stochastic Bidding Algorithm was developed to solve the CDAP. The SBA was proven correct and its completeness and complexity were analyzed. It was shown that the SBA asymptotically solves the CDAP. The average-case complexity is linear with respect to the number of tasks and less than linear with respect to the capability of the agents. This is in contrast to a worst-case complexity that is exponential with respect to the number of tasks and polynomial with respect to the capability of the agents. By using the SBA this constrained nonlinear assignment problem can be solved with the same time complexity as a linear unconstrained assignment problem. The parameter CI, which expresses the effects of clustering, is important to the behavior of the algorithm complexity and should be reduced whenever possible. The number of communication links required increases with the expressiveness with which the problem is modeled. The constraints on task assignments and task schedules should be separated as detailed in this dissertation in situations where the necessary communication links may not be available.

This dissertation presented the Minimum-time Arbitrarily-constrained Distributed Scheduling Problem. The MADSP is for distributed, communicating agents to find an optimal schedule that is feasible with respect to arbitrary predicate constraints. The OptDNSB Algorithm solves the MADSP; the correctness, completeness, and optimality was proven. The OptDNSB Algorithm solves the MADSP through an iterated distributed backtracking technique with an expanding time horizon that yields minimum-time schedules. This also results in a decrease in worst-case complexity in instances where the optimal time horizon is poorly known. The communication network was considered to have an arbitrary topology. It was also shown that task scheduling can be done concurrently with task assignment while the task assignments change without affecting completeness. This is important as it allows the exploitation of the separation between task assignment and task scheduling constraints to operate on an incomplete communication network.

The contributions of this dissertation range from centralized to distributed task assignment and task scheduling. These contributions extend these areas of the literature and provide tools that practitioners can use in automated mission planning. The insights presented here shed light on the connection between task assignment and task scheduling. Specifically that the two can be decoupled for communication benefit. This point of view can be used in mission modeling to assist in mission planning. This work addresses several areas that are relevant to the Air Force and other armed services. These interests concern the exploitation of on-board vehicle intelligence to overcome challenges of operating in an environment where communication may fail and where agents should be able to act independently from a human operator.

5.1 Future Work

Possible improvements related to the modeling of the centralized problem and the Tabu/2-opt heuristic are as follows. Incorporating a more detailed model of ve-

hicle kinematics would result in flyable trajectories that could then be assigned to unmanned aircraft or ground vehicles. Other likely extensions of this work relate to the search itself. The following improvements to the Tabu/2-opt heuristic would provide asymptotic optimality. The Tabu search can be extended to search more extensive perturbations by extending the definition of the neighborhood, $\mathcal{N}_{\mathcal{T}a}$ to include changes in the assignments of more than one task. The 2-opt procedure can also be extended to include k-opt exchanges which would further decrease the optimality gap. Asymptotic optimality results from the fact that considering a perturbation in every task assignment and an n-optimal task schedule results in exhaustive search, in the limit.

Possible improvements related to the solution of the distributed task assignment and task scheduling problem are as follows. It could be shown that the sampling of problem instances used in the computational experiments is uniform. It may be possible to gain computational efficiency by having processes bid on tasks, or clusters, in parallel. Extension of this work to time varying communication graphs, i.e., fault-tolerance, and to dynamic problems, i.e., task execution, are additional areas of future work. With regards to the MADSP and the OptDNSB Algorithm, the work can be extended to include time-varying problems, i.e., where constraints may be removed and not returned, or where tasks may be executed. Another extension would be to use the results of scheduling to improve the dynamic assignment of tasks. This work can also be extended to study how the failure of agents affects the ability to satisfy constraints given that the assignments are decided separately.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Aarts, E., and J. Lenstra (1997), *Local Search in Combinatorial Optimization*, pp. 215–310, John Wiley and Sons.

[2] Adams, J., E. Balas, and D. Zawack (1988), The shifting bottleneck procedure for job shop scheduling, *Management Science*, *34*, 391–401.

[3] Ascheuer, N., M. Junger, and G. Reinelt (2000), A branch and cut algorithm for the asymmetric traveling salesman problem with precedence constraints, *Computational Optimization and Applications*, *17*, 61–84.

[4] Bartak, R., M. A. Salido, and F. Rossi (2004), New trends in constraint satisfaction, planning, and scheduling: A survey, *The Knowledge Engineering Review*, *0*, 1–24.

[5] Bektas, T. (2006), The multiple traveling salesman problem: an overview of formulations and solution procedures, *Omega The International Journal of Management Science*, *34*, 209–219.

[6] Bernstein, D. S., E. A. Hansen, and S. Zilberstein (2005), Bounded policy iteration for decentralized pomdps, in *Workshop on Multiagent Planning and Scheduling*, pp. 52–57, Monterey, California.

[7] Bertsekas, D. (1987), The auction algorithm: A distributed relaxation method for the assignment problem, *Technical report lids*, Massachusetts Institute of Technology, Cambridge, Massachusetts.

[8] Bowring, E., M. Tambe, and M. Yokoo (2006), Multiply-constrained distributed constraint optimization, in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, Hakodate, Hokkaido, Japan.

[9] Brady, R. M. (1985), Optimization strategies gleaned from biological evolution, *Nature*, *317*, 804–806.

[10] Brafman, R. I., and C. Domshlak (2008), From one to many: Planning for loosely coupled multi-agent systems, in *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*, pp. 28–35.

[11] Brafman, R. I., and C. Domshlak (2010), Agents towards vehicle routing problems, in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, Toronto, Canada.

[12] Brandao, J. (2009), A deterministic search algorithm for the fleet size and mix vehicle routing problem, *European Journal of Operational Research*, *195*, 716–728.

[13] Burrows, M. (2006), The chubby lock service for loosely-coupled distributed systems.

[14] Casbeer, D. (2009), Decentralized estimation using information consensus filters with a multi-static UAV radar tracking system, *Phd dissertation*, Brigham Young University, Provo, Utah.

[15] Chandra, T., R. Griesemer, and J. Redstone (2007), Paxos made live - an engineering perspective, in *Annual ACM Symposium on Principles of Distributed Computing*, pp. 398–407, Portland, Oregon, USA.

[16] Chandra, T. D., and S. Toueg (1996), Unreliable failure detectors for reliable distributed systems, *Journal of the Association for Computing Machinery*, *43*.

[17] Cheng, R., M. Gen, and Y. Tsujimura (1996), A tutorial survey of job-shop scheduling problems using genetic algorithms - I. representation, *Computers and Industrial Engineering*, *30*, 983–997.

[18] Cheng, R., M. Gen, and Y. Tsujimura (1999), A tutorial survey of job-shop scheduling problems using genetic algorithms, Part II: Hybrid genetic search strategies, *Computers and Industrial Engineering*, pp. 343–364.

[19] Choi, H., L. Brunet, and J. How (2009), Consensus-based decentralized auctions for robust task allocation, *IEEE Transactions on Robotics*, *25*, 912–926.

[20] Choi, M., and B. Sweetman (2010), The hermite moment model for highly skewed response with application to tension leg platforms, *Journal of Offshore Mechanics and Arctic Engineering*, *132*, 1–9.

[21] Christofides, N. (1976), Worst-case analysis of a new heuristic for the travelling salesman problem, *Report*, Carnegie-Mellon University.

[22] Cordeau, J. F., M. Dell'Amico, and M. Iori (2010), Branch-and-cut for the pickup and delivery traveling salesman problem with fifo loading, *Computers and Operations Research*, *37*, 970–980.

[23] Cordeau, J. F., M. Iori, G. Laporte, and J. J. S. Gonzalez (2010), A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with lifo loading, *Networks Optimization Workshop*, *55*, 46–59.

[24] Dahm, W. J. A. (2010), Report on technology horizons, *Technical Report 1*, United States Air Force.

[25] Dantzig, G. B., and J. H. Ramser (1959), The truck dispatching problem, *Management Science*, *6*, 80–91.

[26] Davis, R., and R. G. Smith (1983), Negotiation as a metaphor for distributed problem solving, *Artificial Intelligence*, *20*, 63–109.

[27] Dewan, P., and S. Joshi (2002), Auction-based distributed scheduling in a dynamic job environment, *International Journal of Production Research*, *40*, 1173–1191.

[28] Dias, M. B., and A. Stentz (2002), Opportunistic optimization for market-based multirobot control, in *International Conference on Intelligent Robots and Systems*, pp. 2714 – 2720.

[29] Dolgov, D. A. (2006), Resource allocation among agents with mdp-induced preferences, *Journal of Artificial Intelligence Research*, *27*, 505–549.

[30] Dumitrescu, I., S. Ropke, J. F. Cordeau, and G. Laporte (2010), The traveling salesman problem with pickup and delivery: Polyhedral results and a branch-and-cut alogrithm, *Mathematical Programming*, *121*, 269–305.

[31] Durfee, E. (1988), *Coordination of Distributed Problem Solvers*, Kluwer Academic Publishers, Norwell, Massachusetts.

[32] Faied, M. (2009), Dynamic mission management and control for adversarial multi-uav operations, Ph.D. thesis, Cairo University.

[33] Faied, M., I. Assanein, and A. Girard (2009), UAV dynamic mission management in adversarial environments, *International Journal of Aerospace Engineering*, pp. 1–10.

[34] Faied, M., A. Mostafa, and A. Girard (2009), Dynamic optimal control of multiple depot vehicle routing problem with metric temporal logic, in *American Control Conference*, pp. 3268–3273, St. Louis, Missouri.

[35] Faltings, B., T. Leaute, and A. Petcu (2009), Privacy guarantees through distributed constraint satisfaction, in *ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pp. 350–358, Sydney, NSW.

[36] Fischer, M., and M. Merritt (2003), Appraising two decades of distributed systems theory research, *Distributed Computing*, *16*, 239–247.

[37] Fischer, M. J., N. A. Lynch, and M. S. Paterson (1985), Impossibility of distributed consensus with one faulty process, *Journal of the Association for Computing Machinery*, *32*, 374–382.

[38] Gendreau, M., A. Hertz, and G. Laporte (1994), A tabu search heuristic for the vehicle routing problem, *Management Science*, *40*, 1276–1290.

[39] Gerkey, B. P., and M. J. Mataric (2004), A formal analysis and taxonomy of task allocation in multi-robot systems, *International Journal of Robotics Research*, *23*, 939–954.

[40] Glover, F. (1989), Tabu search - part I, *Journal of the Operations Research Society of America*, *1*, 190–206.

[41] Golden, B., S. Raghavan, and E. W. (eds) (2008), *The Vehicle Routing Problem: Latest Advances and New Challenges*, Springer, New York, NY.

[42] GuoXing, Y. (1995), Transformation of multidepot multisalesman problem to the standard travelling salesman problem, *European Journal of Operational Research*, *81*, 557–560.

[43] Hall, R. (2006), On the road to integration, *OR/MS Today*, pp. 1–23.

[44] Helsgaun, K. (2000), An effective implementation of the lin-kernighan traveling salesman heuristic, *Technical report*, Roskilde University.

[45] Helsgaun, K. (2009), General k-opt submoves for the lin-kernighan tsp, *Mathematical Programming Computation*, *1*, 119–163.

[46] Hunsberger, L. (2002), Group decision making and temporal reasoning, Ph.D. thesis, Harvard University.

[47] Hunsberger, L. (2002), Algorithms for a temporal decoupling problem in multi-agent planning, in *American Association for Artificial Intelligence*, pp. 468–475.

[48] Hunsberger, L. (2003), Distributing the control of a temporal network among multiple agents, in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*.

[49] Jackson, J., and A. Girard (2011), Distributed task scheduling subject to arbitrary constraints, in *Proceedings of the 18th World Congress The International Federation of Automatic Control*, Milan, Italy.

[50] Jackson, J., A. Girard, S. Rasmussen, and C. Schumacher (2010), A combined tabu search and 2-opt heuristic for multiple vehicle routing, in *American Control Conference*, pp. 3842–3847, Baltimore, Maryland.

[51] Jackson, J., A. Girard, S. Rasmussen, and C. Schumacher (2010), A combined tabu search and 2-opt heuristic for multiple vehicle routing, in *Proceedings of ACC International Conference*, Baltimore, MD.

[52] Jackson, J., M. Faied, P. Kabamba, and A. Girard (2011), Communication-constrained distributed task assignment, *Tech. rep.*, University of Michigan.

[53] Jia, H. Z., A. Y. C. Nee, J. Y. H. Fuh, and Y. F. Zhang (2003), A modified genetic algorithm for distributed scheduling problems, *Journal of Intelligent Manufacturing*, *14*, 351–362.

[54] Kafil, M., and I. Ahmad (1998), Optimal task assignment in heterogeneous distributed computing systems, *IEEE Concurrency*, pp. 42–51.

[55] Kara, I., and T. Bektas (2006), Integer linear programming formulations of multiple salesman problems and its variations, *European Journal of Operational Research*, *174*, 1450–1458.

[56] Karaman, S., and E. Frazzoli (2008), Complex mission optimization for multiple-uavs using linear temporal logic, in *American Control Conference*, pp. 2003–2009, Seattle, Washington.

[57] Karaman, S., and E. Frazzoli (2008), Vehicle routing problem with metric temporal logic specifications, in *IEEE Conference on Decision and Control*, pp. 3953–3958, Cancun, Mexico.

[58] Karaman, S., M. Faied, E. Fazzoli, and A. Girard (2009), Specification and planning of interactive UAV missions in adversarial environments, in *AIAA Guidance, Navigation, and Control Conference*, Chicago, Illinois.

[59] Karp, R. M. (1972), *Reducibility Among Combinatorial Problems, Complexity of Computer Computations (R. E. Miller and J. W. Thatcher, eds.)*, Plenum Press.

[60] Kingston, D., R. Holt, R. Beard, T. McLain, and D. Casbeer (2005), Decentralized perimeter surveillance using a team of uavs, in *AIAA Guidance, Navigation, and Control Conference*, San Francisco, California.

[61] Kirkpatrick, S., J. C. D. Gelatt, and M. P. Vecchi (1983), Optimization by simulated annealing, *Science*, *220*, 671–680.

[62] Knox, J. (1994), Tabu search performance on the symmetric traveling salesman problem, *Computers and Operations Research*, *21*, 867–876.

[63] Krishna, K., K. Ganeshan, and D. J. Ram (1995), Distributed simulated annealing algorithms for job shop scheduling, *IEEE Transactions on Systems, Man, and Cybernetics*, *25*, 1102–1109.

[64] Kuhn, H. W. (1955), The hungarian method for the assignment problem, *Technical paper*, Bryn Mawr College, Bryn Mawr, PA.

[65] Laguna, M., J. P. Kelly, J. L. Gonzalez-Velarde, and F. Glover (1995), Tabu search for the multilevel generalized assignment problem, *European Journal of Operational Research*, *82*, 176–189.

[66] Lamport, L. (1978), *The Implementation of Reliable Distributed Multiprocess Systems*, pp. 95–114, North-Holland Publishing Company.

[67] Lamport, L. (1978), Time, clocks, and the ordering of events in a distributed system, *Communications of the ACM*, *21*, 558–565.

[68] Lamport, L. (1998), The part-time parliament, *ACM Transactions on Computer Systems*, *16*, 133–169.

[69] Lamport, L. (2001), Paxos made simple, Citeseer.

[70] Laporte, G. (2009), Fifty years of vehicle routing, *Report*, HEC Montreal.

[71] Lesser, V. R., and D. D. Corkill (1981), Functionally accurate, cooperative distributed systems, *IEEE Transactions on Systems, Man, and Cybernetics*, *1*, 81–96.

[72] Lin, F. T., and C. C. Hsu (1990), Task assignment and scheduling by simulated annealing, in *Conference on Computer and Communication Systems*, pp. 279–283, Hong Kong.

[73] Lin, S., and B. W. Kernighan (1971), An effective heuristic algorithm for the traveling salesman problem, *Technical paper*, Bell Telephone Laboratories, Inc., Murray Hill, N.J.

[74] Liu, J. S., and K. P. Sycara (1996), Multiagent coordination in tightly coupled task scheduling, in *Proceedings of the Second International Conference on Multiagent Systems.*, pp. 181–188.

[75] Lynch, N. (1996), *Distributed Algorithms*, Morgan Kaufman Publishers, Inc., San Fracisco, California.

[76] Mailler, R., and V. R. Lesser (2006), Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems, *Journal of Artificial Intelligence Research*, *25*, 529–576.

[77] Manathara, J. G., S. B. Sujit, and R. W. Beard (2011), Multiple UAV coalitions for a search and prosecute mission, *Journal of Intelligent Robotic Systems*, *62*, 125–158.

[78] Matsui, T., M. Silaghi, K. Hirayama, and M. Yokoo (2008), Resource constrained distributed constraint optimization with virtual variables, in *AAMAS '06 Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 120–125.

[79] Matsui, T., M. Silaghi, K. Hirayama, M. Yokoo, B. Faltings, and H. Matsuo (2011), Reducing the search space of resource constrained dcops, *Lecture Notes in Computer Science*, *6876*, 576–590.

[80] McAfee, R. P., and J. McMillan (1987), Auctions and bidding, *Journal of Economic Literature*, *25*, 699–738.

[81] Minton, S., M. Johnston, A. Philips, and P. Laird (1990), Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method, in *Association for the Advancement of Artificial Intelligence*, pp. 17–24, Washington D.C., United States.

[82] Moccellin, J. V., and M. S. Nagano (1998), Evaluating the performance of tabu search procedures for flow shop sequencing, *Journal of the Operational Research Society*, *49*, 1296–1302.

[83] Modi, P. J., W. Shen, M. Tambe, and M. Yokoo (2001), An asynchronous complete method for distributed constraint optimization, *Journal of the Association for Computing Machinery*.

[84] Montane, F. A. T., and R. D. Galvao (2004), A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service, *Computers and Operations Research*, *33*, 595–619.

[85] Nagy, G., and S. Salhi (2005), Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries, *European Journal of Operational Research*, *162*, 126–141.

[86] Nissim, R., R. Brafman, and C. Domshlak (2010), A general, fully distributed multi-agent planning algorithm, in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, Toronto, Canada.

[87] Peng, D. T., K. G. Shin, and T. F. Abdelzaher (1997), Assignment and scheduling communicating periodic tasks in distributed real-time systems, *IEEE Transactions on Software Engineering*, *23*, 745–757.

[88] Pezzella, F., G. Morganti, and G. Ciaschetti (2007), A genetic algorithm for the flexible job-shop scheduling problem, *Journal of Intelligent Manufacturing*, *35*, 3202–3212.

[89] Planken, L., M. de Weerdt, and N. Yorke-Smith (2010), Incrementally solving stns by enforcing partial path consistency, in *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling*, pp. 129–136.

[90] Planken, L. R., M. M. de Weerdt, and C. Witteveen (2010), Optimal temporal decoupling in multiagent systems, in *Proceedings of the Nineth International Conference on Autonomous Agents and Multiagent Systems*, pp. 789–796, Toronto, Canada.

[91] Potvin, J. Y. (1996), Genetic algorithms for the traveling salesman problem, *Annals of Operations Research*, *63*, 337–370.

[92] Ren, W. (2007), Multi-vehicle consensus with a time-varying reference state, *Systems and Control Letters*, *56*, 474–483.

[93] Ren, W., R. Beard, and E. Atkins (2005), A survey of consensus problems in multi-agent coordination, in *American Control Conference*, pp. 1859–1864, Portland, OR, USA.

[94] Ropke, S., and J. F. Cordeau (2009), Branch and cut and prices for the pickup and delivery problem with time windows, *Transportation Science*, *43*.

[95] Russell, S., and P. Norvig (2010), *Artificial Intelligence*, Prentice Hall, Upper Saddle River, New Jersey.

[96] Saber, R. O., and R. M. Murray (2004), Consensus problems in networks of agents with switching topology and time-delays, *IEEE Transactions on Automatic Control*, *49*, 1520–1533.

[97] Sadeh, N. (1991), Look-ahead techniques for micro-opportunistic job shop scheduling, *Tech. rep.*, Carnegie Mellon University.

[98] Sadeh, N., K. Sycara, and Y. Xiong (1996), Backtracking techniques for the job shop scheduling constraint satisfaction problem, *Tech. rep.*, Carnegie Mellon University.

[99] Salido, M. A., and A. Giret (2008), Feasible distributed csp models for scheduling problems, *Engineering Applications of Artificial Intelligence*, *21*, 723–732.

[100] Sandholm, T., and V. Lesser (1997), Coalitions among computationally bounded agents, *Artificial Intelligence*, *94*, 99–137.

[101] Sandholm, T. W. (1996), Negotiation among self-interested computationally limited agents, *Phd dissertation*, University of Massachusetts Amherst.

[102] Sarin, S. C., H. D. Sherali, and A. Bhootra (2005), New tighter polynomial length formulations for the asymmetric traveling salesman problem with and without precedence constraints, *Operations Research Letters*, *33*, 62–70.

[103] Schumacher, C., P. Chandler, and S. Rasmussen (2001), Task allocation for wide area search munitions via network flow optimization, in *AIAA Guidance, Navigation, and Control Conference*, Montreal, Canada.

[104] Shima, T., S. Rasmussen, C. Schumacher, N. Ceccarelli, P. Chandler, D. Jacques, B. Kish, and M. Pachter (2009), *UAV Cooperative Decision and Control Challenges and Practical Approaches*, Society for Industrial and Applied Mathematics.

[105] Smith, R. G. (1980), The contract net protocol: High-level communication and control in a distributed problem solver, *IEEE Transactions on Computers*, *12*, 1104–1113.

[106] Smith, S. F., A. Gallagher, T. Zimmerman, L. Barbulescu, and Z. Rubinstein (2007), Distributed management of flexible times schedules, in *Proceedings of*

the second international joint conference on Autonomous agents and multiagent systems, Honolulu, Hawai'i.

[107] Spaan, M. T. J., and F. A. Oliehoek (2008), The multiagent decision process toolbox: Software for decision-theoretic planning in multiagent systems, in *Multiagent Sequential Decision Making (MSDM)*, pp. 107–121.

[108] Stone, P., and M. Veloso (1999), Task decomposition and dynamic role assignment for real-time strategic teamwork, *Lecture Notes In Computer Science*, *1555*, 293–308.

[109] [TCP], Transmission Control Protocol. (1987), Internet request for comments (RFC), *Tech. rep.*, Available from http://www.rfc-editor.org/rfc/rfc1034.txt.

[110] Ullman, J. D. (1975), NP-complete scheduling problems, *Journal of Computer and System Sciences*, *10*, 384–393.

[111] V. Lesser et al. (2003), *Distributed Sensor Networks*, Kluwer Academic Publishers, Norwell, Massachusetts.

[112] Vail, D., and M. Veloso (2003), *Dynamic Multi-Robot Coordination, in A. Schultz et al., 'Multi-Robot Systems: From Swarms To Intelligent Automata, Volume II'*, Kluwer Academic Publishers, Norwell, Massachusetts.

[113] van Laarhoven, P. J. M., E. H. L. Aarts, and J. K. Lenstra (1992), Job shop scheduling by simulated annealing, *Operations Research*, *40*, 113–125.

[114] Wassan, N. A., A. H. Wassan, and G. Nagy (2008), A reactive tabu search algorithm for the vehicle routing problem with simultaneous pickups and deliveries, *Journal of Combinatorial Optimization*, *15*, 368–386.

[115] Weigel, T., et al. (2001), Cs freiburg: Doing the right thing in a group, *Lecture Notes In Computer Science*, *2019*, 52–63.

[116] Wellman, M. P., and W. E. Walsh (2001), Auction protocols for decentralized scheduling, *Games and Economic Behavior*, *35*, 271–302.

[117] Winterstein, S. (1987), *Moment-based Hermite Models of Random Vibration*, pp. 1–29.

[118] Wooldridge, M. (2000), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, The MIT Press, Cambridge, Massachusetts.

[119] Xiong, Y., N. Sadeh, and K. Sycara (1992), Intelligent backtracking techniques for job shop scheduling, in *Conference on Priciples of Knowledge Representation and Reasoning*, pp. 14–23.

[120] Yokoo, M. (1995), Asynchronous weak-commitment search for solving distributed constraint satisfaction problems, *Lecture Notes In Computer Science*, *976*, 88–102.

[121] Yokoo, M., E. H. Durfee, I. Ishida, and K. Kuwabara (1992), Distributed constraint satisfaction for formalizing distributed problem solving, in *IEEE International Conference on Distributed Systems*, pp. 614–621.

[122] Yokoo, M., K. Suzuki, and K. Hirayama (2006), Secure distributed constraint satisfaction: Reaching agreement without revealing private information, *Lecture Notes in Computer Science*, *2470*, 43–66.

[123] Yu, H., R. W. Beard, M. Argyle, and C. Chamberlain (2011), Probabilistic path planning for cooperative target tracking using aerial and ground vehicles, in *American Control Conference*, pp. 4673–4678, San Francisco, CA.

[124] Zhu, X., X. Qin, and M. Qiu (2011), QoS-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters, *IEEE Transactions on Computers*, *60*, 800–812.

[125] Zhuo, H. H., H. Munoz-Avila, and Q. Yang (2011), Learning action models for multi-agent planning, in *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, Taipei, Taiwan.