An Ex-Ante Rational Distributed Resource Allocation System using Transfer of Control Strategies for Preemption with Applications to Emergency Medicine

by

John Doucette

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2012

Abstract

Within the artificial intelligence subfield of multiagent systems, one challenge that arises is determining how to efficiently allocate resources to all agents in a way that maximizes the overall expected utility. In this thesis, we explore a distributed solution to this problem, one in which the agents work together to coordinate their requests for resources and which is considered to be ex-ante rational: in other words, requiring agents to be willing to give up their current resources to those with greater need by reasoning about what is for the common good. Central to our solution is allowing for preemption of tasks that are currently occupying resources; this is achieved by introducing a concept from adjustable autonomy multiagent systems known as a transfer of control (TOC) strategy. In essence a TOC strategy is a plan of an agent to acquire resources at future times, and can be used as a contingency plan that an agent will execute if it loses its current resource. The inclusion of TOC strategies ultimately provides for a greater optimism among agents about their future resource acquisitions, allowing for more generous behaviors, and for agents to more frequently agree to relinquish current resources, resulting in more effective preemption policies.

Three central contributions arise. The first is an improved methodology for generating transfer of control strategies efficiently, using a dynamic programming approach, which enables a more effective employment of TOCs in our resource allocation solution. The second is an important clarification of the value of integrating learning techniques in order for agents to acquire improved estimates of the costs of preemption. The last is a validation of the overall multiagent resource allocation (MARA) solution, using simulations which show quantifiable benefits of our novel approach.

In particular, we consider in detail the emergency medical application of mass casualty incidents and are able to demonstrate that our approach of integrating transfer of control strategies results in effective allocation of patients to doctors: ones which in simulations result in dramatically fewer patients in a critical healthstate than are produced by competing MARA algorithms.

In short, we offer a principled solution to the problem of preemption, allowing the elimination of a source of inefficiencies in fully distributed multiagent resource allocation systems; a faster method for generation of transfer of control strategies; and a convincing application of the system to a real world problem where human lives are at stake.

Acknowledgements

This work could not have been completed without the exceptional efforts of the author's supervisor, Professor Robin Cohen, who has dedicated many a great many hours to the reading and re-reading of its various prototypical forms. Her insightful and direct comments have immensely improved the quality of my work, and I am exceedingly grateful for her contributions.

Professors Peter Van Beek and Peter Forsyth's contributions were also significant. They read and critiqued the thesis in great detail, and provided pointed and useful feedback. I am grateful for all their hard work.

The author gratefully acknowledges financial support from the National Science and Engineering Research Council of Canada (NSERC); its strategic networks of research initiative, hSITE; the Ontario Graduate Scholarship Program; Professor David R. Cheriton and the Cheriton School of Computer Science; The University of Waterloo and its graduate studies office; Professor Malcolm Heywood; and Professor Robin Cohen. Their contributions kept me well fed and housed while pursuing my thesis work, and enabled travel to numerous academic conferences, without which life would have been much less interesting and much less pleasant.

The author further acknowledges mentorship and guidance from many friends and colleagues. Thanks to my officemates, John Champaign and Lachlan Dufton, for keeping the academy lively and for accommodating my appetites for Scandinavian jam holders. Thanks to my classmates, colleagues and friends Atif Khan and Darren Abramson, for their mentorship, hard work, and camaraderie.

A big thanks to my family, especially my spouse Catherine Holloway, and my parents Glenn and Sally.

Finally, thanks to my otherwise unmentioned friends near and far, especially (and in no particular order) Adam Hartfield, Michael Cormier, Travis Bird, Trevor Bekolay, Michael Todd, Patrick Berrigan, Elliot Snow-Kropla, Colin Conrad, Jake Summers, Hadi Houseini and Ahmed Ataullah.

Dedication

For Catherine.

Table of Contents

List of Tables

List of Algorithms

List of Figures

Chapter 1

Introduction

Within computer science, the subfield of artificial intelligence (AI) concerns itself with creating computer programs and systems that are able to eliminate or reduce the need for human cognitive processing. That is, AI attempts to solve high level problems which might normally require a human to reason about the world, or to create elaborate plans. Within AI, a prominent paradigm models AI reasoning through the use of abstracted 'agents' - entities which can both sense their environment in some way, and act to change that environment [53] [59]. Agents can be robots, but more often take the form of a computer program which proposes plans or carries out actions on behalf of human users, while respecting their goals and preferences. A major topic within AI at present is the study of 'multi-agent systems', in which collections of agents with (potentially) different properties co-exist. Agents which act together to solve a problem within a multiagent system are said to be 'cooperative'. For example, a collection of robotic agents might work together to coordinate the exploration of a Martian desert; a group of agents representing investment bankers might, through rational self interest and carefully selected stock market rules, make trades that produce an accurate estimate of a company's value.

Multi-agent resource allocation (or MARA) is a sub-topic within multi-agent systems, wherein the problem to be solved is, broadly speaking, finding an efficient way of allocating a set of resources to a set of agents in order to maximize some function [8]. Most of us encounter problems of this nature on a daily basis. For example, deciding who should sit in the front seat of a car amounts to allocating seats to passengers; planning out the division of labour for a project team amounts to allocating workers to jobs; even obeying traffic laws can be viewed as allocating spots on the street to different cars at different times. Many of these problems are resolved using simple distributed heuristics. For example, we can solve the allocation of spots on the street at intersections by using traffic lights and queues of cars, allocating the resource of the intersection to a car based on whether or not they are in the front of their queue and the light is green in the direction they wish to travel. However, these heuristics do not always make efficient use of resources. If two intersecting streets have very different levels of traffic, then there may be periods when the busy street has long lines of stopped cars waiting at the light while the less congested street has no cars on it at all, leaving an empty (unused) intersectioni[1]. Efficient MARA systems are able to utilize these gaps by dynamically adjusting allocations, as in the autonomous intersection management project of Stone et al. [18][2]. In some environments, the costs of an inefficient allocation can be quite extreme. In hospitals, the scheduling of doctor's time to medical tasks can mean the difference between the life and death of a patient. Although in day-to-day scheduling this is perhaps an infrequent occurrence, mass casualty incidents (MCI), wherein a large number of patients suffer serious injuries at the same time and place [47] pose a greater challenge. For example, MCI are comparatively frequent occurrences in Israel, where suicide bombings and rocket attacks often target vulnerable civilian populations [22]. In these scenarios, the large influx of critical patients may overwhelm a simple scheduling heuristic, like first-in-first-out (FIFO), resulting in unnecessary patient suffering.

The nature of MARA problems naturally suggests several common AI techniques for finding improved solutions. First, adopting a model of the costs and benefits founded in *utilitarianism* [46] (which is the philosophical basis for the prominent artificial intelligence paradigm of utility theory [59]) allows us to formally specify the relative quality of different actions agents could take, and compare the costs and benefit of these actions in a rational manner. Second, work in *decision theoretic reasoning* [33] can allow computational agents to rapidly and precisely compute the utilities of various actions, and select the best alternatives. Third, the incorporation of *machine learning* techniques [3] can allow computational agents to learn from and adapt to changes in their environment. This again serves to increase the flexibility and robustness of such agents to react to rapidly changing circumstances in the allocation problem. Finally, methods from *user modelling* [45] allow computational agents to better serve the needs and constraints of human users when reasoning about the effects of their actions.

Constructing a system using the techniques outlined above might appear intuitively as a good idea, but if such a system is to be practical and useful, it is important to consider precise metrics when evaluating its quality. Consequently, two more formal methodologies

[1]This would happen if, for instance, the green lights (which influence who gets the resource) are merely set to trigger with a certain frequency rather than taking into account traffic conditions.

[2]The project reasons about the demand for an intersection, basing its decisions on the number of cars on the road approaching the intersection, as well as the planned paths and transit times of the cars.

are typically used in assessing the quality of a multi-agent resource allocation system. First, we can use theory to analyze the system and prove properties about it. This approach is used widely, especially by authors like Sandholm [61] [60] [8]. An advantage of this method is that it can allow very broad statements to be made about the system's performance. For example, one might prove that the system will always converge to the optimal resource assignments in a finite amount of time [61]. A corresponding drawback is that these statements are often too broad to provide useful information to practitioners. For example, 10,000 years is a finite amount of time, but a resource allocation method which took 10,000 years to compute its solution would not be especially useful! Consequently, the secondary methodology of simulation is often introduced to evaluate 'typical' performance of a MARA system. Simulations are also widely used within the MARA resource community [51] [8], and are highly complementary to theoretical work. Simulations of a MARA system entail building both the system and a virtual approximation of the environment in which the system is intended to function. By running the system within this virtual environment, it is possible to glean information about the behaviors of the system in the real world. Unfortunately, the results of simulation are contingent on the virtual representation of the world incorporating all salient properties of the environment. Although this is impossible to determine without genuine empirical benchmarking of the system, simulation can still provide valuable insight into whether a system has even the potential for practical use.

The principal problem addressed in this thesis is the issue of 'preemption cycles' in resource allocation tasks. Preemption cycles are a special case of a cyclic dependency in a resource allocation task. Challenging cyclic dependencies arise when agents are 'myopic', meaning that they do not consider the long term consequences of their actions, and will only change their state if they see an immediate benefit in doing so[50].

For example, consider figure 1.1. Imagine that Alice, Bill, and Clara are patients in a hospital where doctors have agreed to use an automated scheduling system. Bill and Clara have doctors treating them right now, and Alice does not. Alice's agent attempts to compute the relative costs and benefits of insisting that Bill yield his resource to Alice. The benefits are easy: Alice gets the resource, and will be treated sooner. The costs however, can become complex. As a worst case bound, a conservative approach might choose to estimate the costs as being equivalent to the loss in utility from delaying Bill's treatment until after Alice is done. Unfortunately, this might prevent valid preemptions from taking place. For example, if there is another doctor who is available to treat Bill, then this fails to consider the other actions Bill's agent could take following the preemption, A better estimate of the cost associated with taking Bill's doctor is then to assume Bill will simply be 'bumped' to this new, available doctor. We therefore propose considering what we refer to as the *opportunity cost* of the preemptions - i.e. the cost, considering

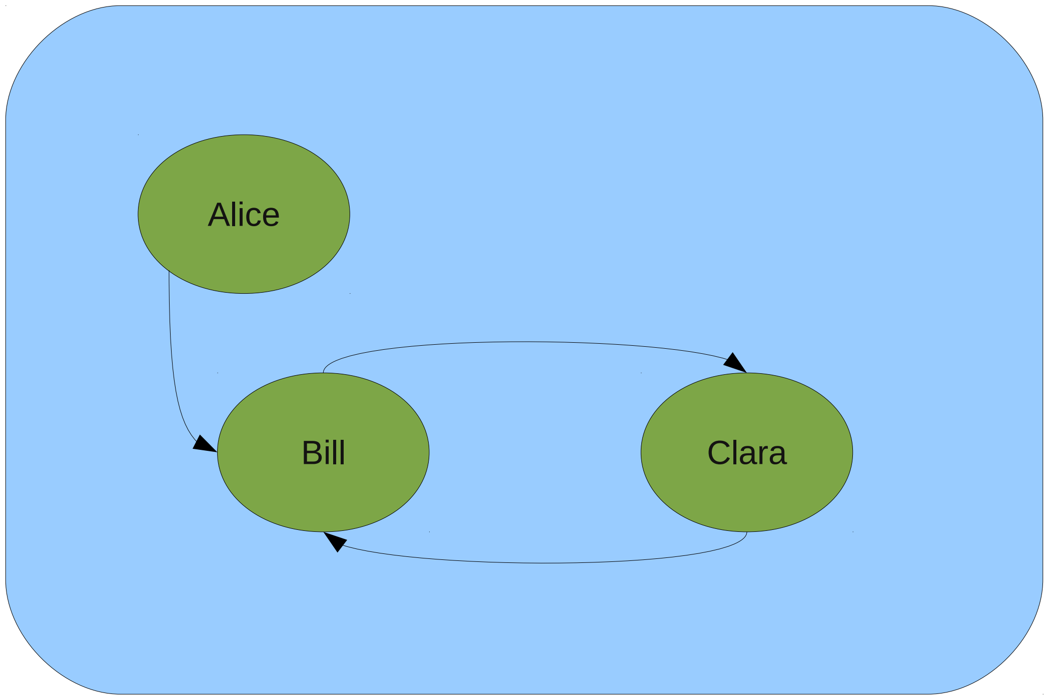


Figure 1.1: Example of a preemption cycle.

alternate allocations of resources. Continuing with the example above, suppose that there is no doctor who is free, and that the only available doctor is the one treating Clara. The opportunity cost of preempting Bill's doctor now depends on whether Bill's agent can, in turn, preempt Clara's doctor. Coincidentally Clara's opportunity cost for this action may depend on the possibility of her being able to obtain Bill's doctor, forming a cyclic dependency. Computing an exact estimate of the costs is now non-trivial, and failure to resolve the dependency will result in an incorrect opportunity cost estimate, which might prevent Alice (the original preemptor) from receiving care. This is a preemption '2-cycle'. If we had a longer chain of cyclic dependencies, in which n agents were cyclically dependent, we would have a preemption 'n-cycle'.

Ultimately, algorithms for resolving MARA problems that allow for preemption need to specify how cyclic dependencies can be addressed. The conservative approach suggested above, where the cost simply refers to what happens if the preempted agent is left without a resource at all, is therefore inappropriate, and can end up causing dramatic and needless inefficiencies. Consider figure 1.2. Patients Bob, Joe, and Charlie are in a hospital, where, once again, doctors have agreed to use an automated scheduling system. The patients each need to be treated by a specific deadline shown beneath their names. Bob is in the most critical condition and must be treated immediately, within 1 time step (notation (1)), or he will die. Unfortunately, Bob has arrived last and is initially placed at the back of a queue for treatment. Under ordinary circumstances, Bob would easily preempt whoever was at the front of the queue with his critical condition. However, in this case Joe is not quite as badly injured as Bob, but still needs to be treated very soon (within two time steps (2)). When Bob attempts to preempt Joe's timeslot (the only one which will save his life), Joe's agent overestimates the damage this will cause by failing to consider the preemption of Charlie's time slot, and instead considering only the nearest free time slot, 4 steps away. Consequently, Joe's agent will reject the exchange, needlessly killing Bob. If Joe's agent were less myopic and could consider the costs of preempting Charlie's timeslot without worrying about the development of cyclic dependencies, then the arrangement (Bob, Joe, Charlie) could be discovered, and all patients could be saved.

Various methods have been used to mitigate the preemption cycle problem, but present solutions are not wholly satisfactory. For example, Paulussen et al. [51] made agents slightly less myopic by allowing the first agent in a preemption cycle to perform a single-step lookahead. However, subsequent agents in the cycle were restricted to myopic worst case bounds on the costs of their preemptions, and could not reason about acquiring additional resources following the preemption. Paulussen's approach can resolve the simple cycles presented in the examples above, but cannot resolve cycles of length 3 or greater because even a single step lookahead will not discover any *pair* (or larger set) of transfers that can produce a net improvement in utility (a longer lookahead is required). The details of several prior approaches to the problem, and a more extensive discussion of their limitations can be found in chapter 4.

The primary contribution of this work then, is the creation of a novel approach to the problem of estimating preemption costs for the purpose of proposing solutions to MARA problems that allow agents to be less myopic. We accomplish this through a careful combination of planning and coordination techniques, which are discussed in chapters 3 and 5. Additionally, we apply a new system based on these techniques to resolve a resource allocation task in a congested medical domain, discussed in chapter 6. Our new approach contains two important pieces: better estimates about the opportunity costs of agents' actions, and a Rawlsian [56] approach to the philosophical problem of distributed justice, which yields a *semi-coercive* preemption policy. This means that when resources are

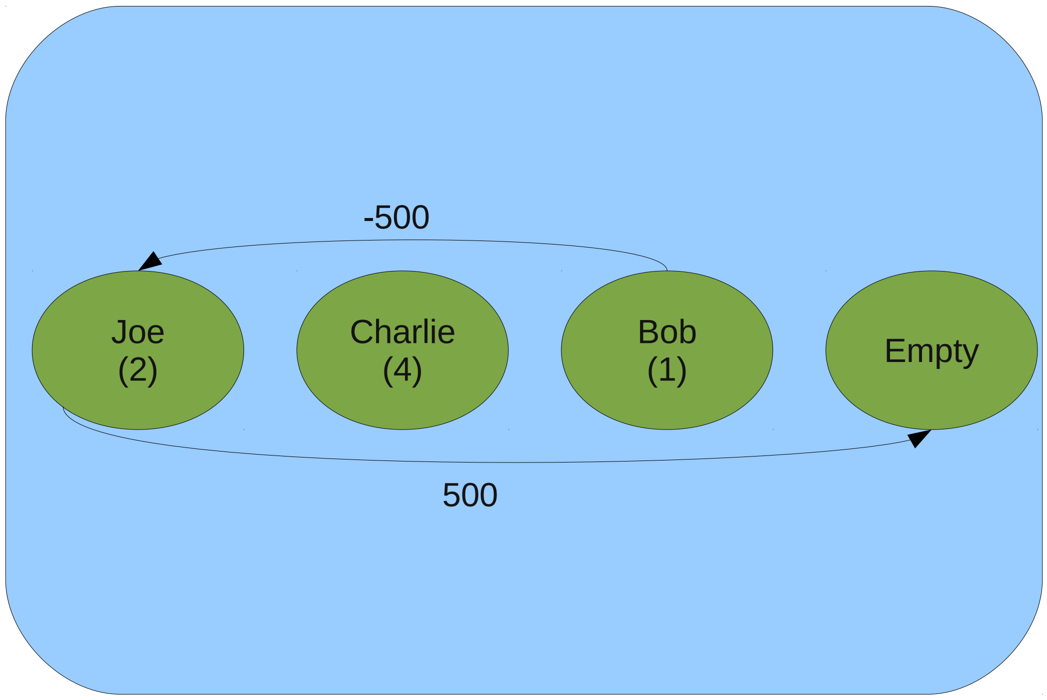


Figure 1.2: A possible MCI schedule where avoiding opportunity cost estimation may over estimate the costs of a resource preemption. Patients will be treated in order from left to right, and will die if not treatment begins any later than the number given in their bubble. While an arrangement exists which will allow all patients to live (Bob, Joe, Charlie), Joe's agent will report Joe's death as the cost of preemption, because the nearest timeslot certain to be available is the empty slot, 4 units back. As the benefits of the preemption (1 life saved) are no greater than the system's estimate of the costs (1 life lost), the scheduling system incorrectly concludes that no improvement is possible and prevents the preemption. Note that we equate a life with a value of 500 units.

preempted, good justifications (ones based on utility metrics) must be provided before the preemption can proceed, in contrast to a fully-coercive system where an agent can preempt a resource solely on the basis of wanting it, without providing justification for why its needs might exceed the needs of the agent who holds it. As our estimation of opportunity costs entails the use of computationally expensive 'Transfer-of-Control' strategies (discussed in chapters 2 and 3), we also offer a secondary contribution in the form of a detailed analysis of algorithms for generating such strategies in the context of resource allocation. The concept of a Transfer of Control strategy arises in the field of adjustable autonomy multiagent systems [29], and is used to allow agents to reason about a series of alternate allocations of (tasks or) resources to entities into the future. It is this 'forward-looking' reasoning that ultimately allows agents to be less myopic about the resources they may obtain following a preemption, resulting in allocations with better overall utility. Our system is demonstrated in the context of a medical scenario known as a 'Mass Casualty Incident', where centralized scheduling algorithms could experience especially great difficulty. We compare our approach to that of previous authors in the medical domain [51] [50] [14]. In particular, we run simulations allocating doctors to a large number of incoming patients, and are able to show that our methods produce dramatically fewer patients whose health states deteriorate to a critical condition. Our solution also integrates learning techniques, the value of which we are able to quantify in our simulations.

The remainder of this document is organized into 7 chapters. The reader is encouraged to begin in chapter 2, which contains much of the background information necessary for a full comprehension of the thesis. Chapter 3 presents a detailed discussion and comparison of several algorithms which can be used to generate transfer of control strategies more or less efficiently. Chapter 4 is concerned with the abstracted issue of preemption costs in general, and clarifies the term "Ex-Ante Rationality", while chapter 5 is concerned with the formal specification of our solution to the problem in terms of models and algorithms. They should be read after the background material, but before chapter 6, which demonstrates an application of the completed system for assigning doctors to patients during a mass casualty incident. The document concludes with chapters 7, and 8, which present a discussion of the results from earlier chapters, and some interesting avenues for future research respectively.

Chapter 2

Background

This chapter is intended to provide the reader with a somewhat detailed summary of the knowledge required for proper comprehension of the thesis. Several topics are covered at length. We begin with a more extensive description of Agents, Multiagent Systems, and Multiagent Resource Allocation (MARA) problems, which are the central focus of the work, and especially their appearance in medicine. This is followed by a detailed presentation of Transfer of Control Strategies, which form a major component of the MARA system we will propose. We then discuss bother cost modeling, which is utilized both for the modeling of human resources and as inspiration for certain learning components in our system. Finally, we conclude with a description of Mass Casualty Incidents (MCI), which are utilized as a case study of the effectiveness of the new system.

2.1 Agent Centric Artificial Intelligence

Within the field of Artificial Intelligence (AI), a current and important framework for modelling the behavior of AI systems is the idea of an 'agent'. Agents are broadly viewed as any system which is capable of both sensing its environment and acting to alter it in accordance with some goals [59].

Of greater interest to modern AI researchers is the notion of a 'computational agent', that is, a software program which acts in furtherance of some (human specified) goal. Russell and Norvig [59] further subdivide such agents into a number of subtypes, but we are concerned only with one in this work, the 'utility-based agent'. A utility-based agent is one which has some internalized notion of desires, where complex sequences of actions and desires can be implicitly encoded in a specification of preferences. For example, if I tell a utility-based agent that I would like to have a television, and that having one is worth \$50 to me, then the agent can take actions which not only acquire a television, but acquire one for the lowest price possible (maximizing my economic surplus). These agents are useful in the domain of resource allocation because a specification of preferences over different kinds of resources allow a utility-based agent to reason about the relative values of different possible actions.

Agent Centric AI has become closely tied to the idea of a 'multiagent system': a community of autonomous, rational agents who interact with one another in the pursuit of a common goal. A number of paradigms are adopted through which the agents may interact [82]. In the centralized paradigm, the agents simply report their preferences and/or capabilities to a central authority who then decides how best to solve the problem using the agents. In the cooperative paradigm, agents coordinate with one another to solve the problem without help from a centralized authority. For example, they could broadcast their capabilities or preferences to one another and then make local decisions about what to do. Finally, in the competitive paradigm, problems are solved as a byproduct of competition between agents. For example, agents asked to determine a fair market price for an object could engage in a series of speculative, but self-interested trades, which, if the agents are rational, will reach equilibrium at the fair market value of the object.

Multiagent systems have been applied to a large number of real world problem domains. They are often a natural method for modelling the behaviors of people in real world marketplaces like eBay [57], and competitive profit maximizing agents are used to solve logistic problems in travel markets [79]. In security settings, theories and systems from multiagent systems research are used to secure Los Angeles International Airport [52], and for emergency and disaster response [66]. For resource allocation, applications include the scheduling of medical patients [51], sharing of time on a multi-national satellite [8], and the design of futuristic intersections for autonomous cars [18].

Within multiagent systems, it is this last topic, multiagent resource allocation, which forms the focus of this thesis.

2.2 Multiagent Resource Allocation

2.2.1 What is a Resource Allocation Problem?

Multiagent resource allocation problems constitute the central topic of this thesis, but to better understand them, we will first consider the broader topic of resource allocation problems in general.

A resource allocation problem is essentially an optimization task [82]. We are given a set of resources R and a set of entities A. Each entity has some preferences about the resources they would prefer to receive, which are represented by a utility function $U : R \times A \rightarrow \mathbb{R}$. Given a particular resource and entity, the utility function produces a quantity of utility indicating how pleased that entity would be if it were assigned that resource. The objective of the problem is to find an assignment of resources to agents that maximizes some social welfare criterion. Naturally the criterion we choose to maximize can dramatically alter our choice of allocation. For example, we might adopt an egalitarian criterion which tries to ensure that agents receive resources of comparable value, or a utilitarian criterion which simply maximizes net utility even if this means dramatically benefiting one agent over the others.

Common social welfare criteria [8] include:

- Utilitarian social welfare, which maximizes the sum of the individual utilities of all the agents. If r_e is the resource assigned to entity e, then the utilitarian social welfare function for an assignment is given by

  $$sw_{util} = \sum_{e \in A} U(r_e, e)$$

  This method has the nice property of being 'efficient': if the utility function is an accurate representation of the relative values of different assignments, then no other criterion can produce a larger amount of utility than this one. Unfortunately, utilitarian social welfare places no limits on the equality of the resulting allocation, allowing some agents to receive much larger value than others.

- Egalitarian social welfare addresses the issue of equality by explicitly optimizing the minimum amongst all agents' individual utilities. It is given by:

  $$sw_{egal} = \min_{e \in A} U(r_e, e)$$

  This method only partially addresses the issue of fairness however, since the relative value of different allocations with the same minimum utilities is unspecified. Highly unfair allocations where, for example, one agent receives 1000 utility and everyone else gets 5, would be preferred to other allocations where all agents but one receive 500 utility, with the final agent getting nothing. Thus, the function imposes a tyranny of the individual over the group, ensuring that no one person is exploited to achieve better outcomes for others.

U	d_1	d_2	d_3
Bob	4	2	2
Margaret	8	1	5
Susan	0	3	1

Table 2.1: An example utility function for resource allocation

- The Nash Product is a social welfare measure which mitigates issues of equality somewhat better by maximizing the product of all the agents' individual utility functions rather than the sum. Using the notation from utilitarian social welfare above, this is given by:

  $$sw_{Nash} = \prod_{e \in A} U(r_e, e)$$

  with the added constraint that individual utilities must always be non-negative. A nice property of the Nash product is that optimizing it will preferentially provide gains to many agents (since improving the smallest value in the product yields a much larger increase than improving the largest by the same amount), but will still prefer some inequity if it provides great benefits to certain members of the society.

With the information above, we can specify an example problem. Let us set R to be a set of doctors, and A to be a set of patients. Patients will have preferences about which doctors treat them, presumably based on a combination of the doctor's individual specializations and bedside manners, which we can capture in a utility function U. More concretely, we suppose that $R = \{d_1, d_2, d_3\}$, $A = \{Bob, Margaret, Susan\}$ and U is given in table 2.1

If we want to maximize utilitarian social welfare, and we assume each patient needs only one doctor, then we would assign d_1 to Margaret, d_2 to Susan, and d_3 to Bob, for a total of:

$$sw_{util} = U(d_1, Margaret) + U(d_2, Susan) + U(d_3, Bob) = 8 + 3 + 2 = 13$$

If we wanted to maximize the Nash Product however, it would be better to assign d_1 to Bob, d_2 to Susan, and d_3 to Margaret, giving us:

$$sw_{Nash} = U(d_1, Bob) * U(d_2, Susan) * U(d_3, Margaret) = 4 * 3 * 5 = 60$$

instead of the $8 * 3 * 2 = 48$ achieved under the utilitarian social welfare optimum.

In the context of this work, we are primarily interested in allocations which maximize utilitarian social welfare.

2.2.2 How are Resource Allocation Problems Solved?

Resource allocation problems can be solved in a number of different ways. Following Ye et al. [82], we elect to present these methods along a continuum of 'multiagent-ness'. Methods which are more mutliagent-like will tend to rely on the autonomy of the entities in the problem to find a good allocation. Methods which are less multiagent-like will tend to treat entities more like inputs to an optimization problem, and may use some other non-agent based method to actually find a good solution.

We note before proceeding that, whatever methods are utilized, resource allocation is an NP-Complete problem [9], and so finding optimal solutions to generalized problems is not likely to be easy. For this reason, our discussion of the various methods advantages and disadvantages will tend to focus on alternative criteria, including communication overhead, incrementalism, and coercion.

At one extreme are the full centralized optimization approaches, which include techniques from optimization and operations research. A good example of such a technique consists of phrasing the problem as an integer linear program [13] (which is typically quite straightforward), and then solving the program using, for example, the simplex method [13] coupled with branch and bound search [41]. An advantage of this approach is that it is simple to construct and deploy, which has led to widespread adoption in resource allocation tasks, including the assignment of shifts to nurses [34] operating theaters to surgery teams [4] and money to hospital projects [38]. Other centralized techniques include heuristic guided search, which typically uses a problem-specific heuristic to efficiently locate good (but not necessarily optimal) allocations [54] [55].

There are certain essential drawbacks to the utilization of a centralized approach. First, centralized approaches require a very large amount of information to be aggregated and stored in a single place. Each entity must make its preferences known to the centralized authority, and any changes in an entity's preferences must be registered with the authority. Further, failure in or congestion of communication channels to the authority can render the system useless, requiring a high degree of reliability in the underlying infrastructure. Second, centralized approaches often lack incrementalism. If an entity changes its preferences half way through the optimization of a good solution, the algorithm may need to start anew, rather than proceeding from its new state. Finally, centralized approaches are always in some sense coercive, since the centralized authority ultimately decides who should and should not receive different items. That said, they may also be considered non-coercive insofar as rational agents may still be persuaded to accept the authority's judgements under certain circumstances.

Moving up the spectrum, we consider cooperative multiagent methods, in which entities are assigned agents who coordinate to find a good allocation [82]. Often this work is phrased in terms of coalition formation [60], where in agents form ad-hoc teams in order to maximize their individual gains.

Finally, in competitive approaches, entities are assigned to agents who negotiate on their behalf in order to maximize the entity's individual utility, even at the expense of producing a poor allocation overall. An important result in this area is that rational, self-interested agents who make only monotonically increasing swaps of resources, with monetary compensation if necessary, can always eventually find solutions which maximize social welfare [61]. Unfortunately, this result does not hold in the case where agents cannot exchange money.

Both above multiagent methods offer advantages in terms of communications reliability and efficiency (since they are distributed), incrementalism (many such methods can engage in localized negotiations to address localized changes in the problem domain), and coercion (agents are usually partially or wholly autonomous, meaning that they are often free to choose whether or not to give up held resources).

2.2.3 Evaluation

To be applicable in the real world, a resource allocation technique should have a number of desirable properties, which we take in part from Sandholm's study of the topic [61]. First, the technique should be computationally tractable, so that it can be fully realized. Second, it should be distributed, meaning that it does not require a centralized authority to aggregate information, and thus, does not have centralized points of failure. Third, the technique should be non-coercive, meaning that people would voluntarily elect to use it. Finally, the system should be efficient, meaning that, at minimum, it offers some improvement over simple heuristic approaches, and preferably over complex existing allocation techniques as well.

Notably, we do not consider Sandholm's fifth criterion, stability, in the context of our work. Stability is often a desirable property in, for example, an auction-style setting where agents might wish to continue buying and selling until a stable equilibrium has been reached where everyone is, in some sense, happy with their allocation. In this work however, we are more concerned with resource allocation scenarios where allocations are likely to change. That is, scenarios like a hospital where resources (e.g. doctors) might come and go, and entities are entering and exiting the problem at arbitrary points in time (as existing patients are cured and new ones arrive). In such a scenario, stability is unlikely to occur. Further, even if a stable solution can be achieved, it is unlikely to be stable for long. In light of this, we prefer to examine the ability of a system to cope with sudden 'shocks' or perturbations, and optimize for robustness to these rather than stability *per se*.

2.3 Transfer of Control Strategies

Transfer of Control (TOC) strategies were developed as a planning technique for use in multi-agent systems like the Electronic Elves project [63]. They are used in 'mixed-initiative' multiagent systems [49] [29], where both humans and computational agents are able to act, and in systems where agents have 'adjustable-autonomy' [65] [44], that is, where agents have the capacity to offload their decision making to other entities within the system.

In the Electronic Elves framework [63], a transfer of control operation is an attempt by an agent to transfer decision making control to another entity within the system. For example, an agent might be tasked with deciding when to hold a meeting at which a particular group of people must be present. Each of these people have their own preferences and constraints, and the agent needs to pick the time which is most likely to satisfy them. If the agent is unsure about which times may be best, it can transfer control to one of the people who needs to be at the meeting, effectively asking them to pick a time instead. Naturally, some people are more likely than others to pick a reasonable time, and the agent can take this into account when deciding who to ask.

Transfer of control strategies are lists of entity/time pairs, which describe a sequence of TOC operations. The need for strategies arises because the first person selected by the agent to make the decision may decide that they cannot make it, or might simply ignore the agent entirely. In order to ensure that the decision is made, the agent reasons about a sequence of actions which maximize the probability of a good decision being made. For example, the agent might transfer control over the decision to a manager during the 9-5 work day, but, if the manager fails to make the decision, the agent takes back control. The agent could then transfer control to a different employee (say, a night manager), or just make the decision itself [62]. A sample TOC strategy might look like $\{\{e_1 t_1 e_2 t_2 A\}\}$, representing a transfer of control to entity e_1 initially (i.e. asking e_1 to take over the current decision making), then to entity e_2 at time t_1 if e_1 has declined to respond, and then having the agent (A) make the decision itself if necessary at time t_2.

The challenge then is to determine the best choice for the entities (e_i's) and times (t_i's), typically to maximize some overall expected utility. This utility is estimated by modeling factors such an entity's likelihood of response and expertise in handling a task (and thus the expected utility if that entity is making the decision). Finding the optimal times to switch between two given entities consists of finding the point in time where the expected value of transferring control to the new entity exceeds the expected value of keeping the current entity in control. We can accomplish this by finding the nearest point at which the two functions intersect [63].

Cheng [7] extended TOC strategies to incorporate *partial* transfers of control which first ask entities questions, rather than simply transferring control of decision making to them, as in the EELVES framework [63], with this older operation being renamed a 'full transfer of control'. Thus, for example, an agent could begin the process by asking a manager who should make the decision, rather than simply assigning the decision to the manager, or by asking an entity whether they have enough time to make the decision right now.

'Generating' transfer of control strategies is the process of selecting the best possible sequence of full and partial transfers of control in order to produce a high quality decision. This process reduces to an optimization problem, and is typically solved using branch-and-bound search heuristics [5] [63]. The optimization proceeds by maximizing the integral of the expected quality of the decision made by the entity currently in control of decision making, multiplied by the probability that the entity will actually make the decision at the current time:

$$EU = \int_0^\infty P_T(t) * EU^d_{e_{current}}(t)\ dt$$

For example, if the expected utility of a decision made by entity 1 in the example TOC strategy above is 5, and the probability of entity 1 making the decision before time 1 is 0.4, then the expected utility of starting a TOC strategy with e_1t_1 is $0.4 * 5 = 2$. Partial TOCs are handled by applying a weight to the expected value of the TOC operations that would be taken given each possible response, equal to the probability estimate for that response being produced.

TOC strategies are a useful decision making tool because they tell us who should be making a decision, and how long we should wait for them before trying someone else. As we shall see later in the thesis, this process closely parallels the process of finding a good resource to address a particular need, allowing TOC strategies to play an important part in multiagent resource allocation systems.

2.4 User Modelling

A final, but important, set of tools utilized in this thesis come from the AI subfield of user modelling.

User modelling is concerned with modelling the behaviors or intentions of particular humans that an AI system needs to interact with [23]. For example, while some users might find novice hints about a new piece of software useful, others might find them irritating. A system which takes this into account and builds an internal representation of its user could present hints only to those who need them.

Of particular concern in this work is the minimization of 'bother costs'. Bother costs arise when a system actively seeks the user's attention. For example, if the system calls a user's cell phone to notify them about an alert, it is bothering the user, producing an associate (potential) decline in utility. Naturally, a system which can model the costs associated with bothering the user and which can therefore minimize said costs, can offer significant advantages over one which lacks this feature. For example, a system that can recognise that its user is asleep might decide to delay less important alerts until the morning.

Fleming [25] proposed a bother cost model based around an exponentially decaying sum of the bother experienced by the user in the past. The model had two central components. First, the bother experienced by a user so far was given by:

$$BSF = \sum_{i \in I} c(i) * \beta^{t(i)}$$

where I is the set of all previous interactions between the system and the user, $c(i)$ is a measure of the bother caused by a previous interaction i, $t(i)$ is the time at which that previous interaction took place, and β is a user-specific discounting factor, which is used to model the decay in bother caused by past interactions. Essentially this models a user who forgets the annoyance of previous interactions at a rate determined by β. When a new interaction takes place, the cost of the interaction $c(i)$ will be dependent on this sum, leading to an exponential rise in the cost of closely spaced interactions. This represents a user model where users are happy to tolerate interactions as long as they are spread out, and the system does not bombard them with a large number of interactions at once. The second component was the bother increase function, which was a user-specific measure of tolerance for bother. This function produced the values for $c(i)$ in subsequent interactions, and tended to rely on BSF, making it some form of exponential.

Cheng [5] expands upon this by incorporating the complexity of an interaction based on Bauer et al.'s model [1], and the "attentional state factor", based on Horvitz et al.'s work, [32] into the cost of an interaction. The former allows for differentiation between, for example, a request to indicate a preference and a request to look up detailed information in determining the bother caused to a user. The latter allows modeling of how disruptive an interaction is to a potential user, allowing, for example, a user currently hard at work or asleep to be modeled as experiencing greater bother than one who is currently taking a break. Jung [35] applied the framework of Cheng [5] to the problem of medical resource allocation, modelling the bother costs of doctors by using skill sets to estimate user unwillingness factors, and the busyness of the doctor to estimate attentional state factors. A detailed discussion showing the application of these methods to a specific problem appears in chapter 6.

Bother cost models can be particularly useful when we are coordinating the interactions between agents and humans in a mixed-initiative multiagent system. This is because, even if a particular agent only bothers a human once every couple of hours, a system containing hundreds of such agents may bother humans to a much greater extent, in a sort of 'tragedy of the commons'. Cohen et al. [5] consider a number of possible methods to address this. In particular, the 'Type IV' proxy agent they propose offers a considerable advantage in such problem domains. Type IV proxy agents are special agents assigned to each human in the system. The agent maintains an internal bother model of the sort discussed above, representing the human to which it is attached. When another agent wishes to contact the human, it must first clear its request with the proxy agent in order to verify that its information about the bother experienced by the user is not out of date (i.e. that the information is within a threshold of the actual bother as calculated by the proxy).

Bother costs models of this sort are of interest in the context of this thesis for two reasons. First, as we shall discuss shortly, MARA problems often involve human actors. For example, in medical resource allocation doctors might take on the role of resources within the system. Each time the system wants them to do something else, an interaction must occur. Being people, doctors possess some (individualized) threshold for the frequency of interactions before they will begin to ignore them. For example, in one study where a computerized system could alert doctors when a prescription posed a danger to the patient, 50% of doctors chose to reduce the number of alerts they received, and only 10% of alerts were examined in detail [74]. Doctors ignore many of these alerts because the benefits outweigh the costs, or because no alternative medication exists [77], and report "that the sheer volume of alerts interferes with workflow, increases the likelihood that they will fail to respond to critical prescribing problems, and creates substantial barriers to the use of electronic prescribing systems altogether" [74]. For these reasons, we would like a MARA system, especially a MARA system for use in medical settings, to be able to model the bother costs of its interactions.

Second, these bother cost models represent a simple class of learning algorithms which we can use to model other forms of congestion. As we shall see later in the thesis, this can be a useful ability for agents in a MARA system to possess.

2.5 Mass Casualty Incidents

Finding good or optimal allocations of doctors to patients in a "Mass-Casualty Incident" (MCI) is an especially interesting MARA problem, which is considered in some detail during the course of this thesis. MCI occur when a large number of people unexpectedly suffer serious injuries at the same time [47], as might happen in a mass transit accident or in a terrorist bombing. MCI should not be confused with "large scale emergencies" like floods, influenza pandemics or wars, which also produce large numbers of casualties, because such events typically occur both on a larger scale and over a longer timespan, and are likely to be anticipated by hospitals to some extent. Recent analysis of MCI events in Israel (mainly suicide bombings), indicates that most victims of MCI events in regions of frequent occurrence are triaged and transported to a hospital in an extremely short time period (15-20 minutes). Unfortunately, this efficiency comes at a cost. Patients are overwhelmingly more likely to be evacuated to the hospital nearest to the incident[1] [22] than to any other. As the proximity of the hospital is no guarantee of its suitability for the task at hand, this can produce problems. The nearest hospital is unlikely to be a dedicated trauma centre, and staff may not be equipped to deal with a large influx of emergency patients. More worryingly, in typical incidents most patients are not transfered to a dedicated trauma centre after arrival at a nearby hospital, possibly due to the pride of hospital staff [31]. Thus the problem posed is as follows: Given an "ordinary" hospital (i.e. not a dedicated trauma centre), with a diverse set of medical personnel, some of whom may not be extensively trained in emergency medicine, treat a large and sudden influx of MCI patients so as to minimize cost and patient suffering.

There are two distinct problems for consideration in this domain. First, the efficacy of triage systems in MCI is questionable, as shown in recent work examining the specificity and sensitivity of the widely used START system [36]. Analysis also suggests that the unique nature of injuries sustained by victims of MCIs may necessitate the use of specialized MCI systems, as patients may have secondary conditions which warrant attention but are not immediately obvious. As an example, in suicide bombings, some victims will experience minor injuries in the form of human remains shrapnel [31]. While an ordinary triage

[1]The odds ratio reported for this association is a staggering 249, an effect 25 times stronger than the next statistically significant factor!

system would mark these patients as low priority, this type of injury carries increased risk of infection, and might receive higher priority [31]. Second, the allocation of doctors and other medical staff to properly triaged patients is also a non-trivial problem because additional patients continue to arrive (in smaller numbers) for as long as 2 hours following the incident [22]. For this reason, some surgical teams are often held in reserve working on less urgent cases to ensure that if the most serious patients arrive somewhat later than most victims, they can still be treated promptly [21]. This work puts more emphasis on the latter issue, and proposes a framework for multi-agent resource allocation in hospitals which is tailored to handle mass causality incidents.

Chapter 3

Efficient Computation of Optimal TOC Strategies

3.1 Introduction

Before plunging into the principal topic of the thesis - distributed resource allocation with preemption - we examine a related issue investigated at some length in pursuit of the former. The topic of interest in this chapter is the *efficient* generation of *optimal* TOC strategies. Thus, in this chapter we consider several possible ways in which a single agent could reason about the optimal series of interactions with other entities to accomplish a particular task. For example, an agent who wishes to schedule a meeting might be able to contact any of hundreds of company employees in order to determine preferences, or simply assign the decision to a knowledgeable party. When our agent attempts to contact an entity, the entity may be busy or unavailable. If the decision is assigned to an entity, that entity might elect to refuse to take it on, leaving the agent to make the decision or assign it elsewhere. Finally, some entities might make better decisions than others, which renders the agent's choice a weighty matter. We refer to an agent's attempt to contact an entity as a transfer of control (TOC) request or operation, or occasionally simply as a 'TOC'. Our agent's goal then amounts to finding the sequence of TOCs which is most likely to produce a good decision: an optimization problem.

The optimization algorithm we use to solve this problem, as mentioned above, needs to be both very fast and highly accurate. This is because the algorithm will be used extensively in the system which forms the primary topic of this thesis. A very large number of such strategies will need to be generated in order to find an optimal order, necessitating an efficient algorithm. Further, the efficiency cannot come at the expense of accuracy, as might be accomplished through the use of a heuristic approximation. The problem is rife with co-dependencies, which facilitate the propagation (and thus, magnification) of any errors in assessment. In light of this, a dynamic programming based approach to TOC strategy generation is investigated. Promising theoretical properties are proven, and a comparison with competing heuristic search methodologies is conducted.

3.2 Problem Description

Formally, TOC strategy generation is described as a temporal optimization problem. The optimizing agent is provided with:

1. A set of possible TOC operations O. This set may be partitioned into Full (asked to make a decision) and Partial (asked a question) transfers of control, as described in chapter 2, and in [5]. We refer to these subsets as F and P respectively. We use terminology “Agent f receives control” and “Agent f makes the decision” to denote the execution of a TOC operation which gives control to an agent or entity f, as a shorthand.

2. A time step size Δt, which describes the resolution of the search. A smaller Δt may allow higher quality TOC strategies, but will also increase the search space.

3. A maximum search depth k, which describes the maximum number of time steps (in units of Δt) that can comprise a valid solution to a TOC strategy in this context. k may be thought of (informally) as the maximum number of agents to which control can be transfered during the resulting strategy (a unique agent for every time step), although it is important to note that the *minimum* number of agents in a strategy can always be lower than k (i.e. if we transfer control to the same agent twice).

4. A Valuation Function $V : O \times \mathbb{Z} \to \mathbb{R}^+$, a function which describes the expected utility of a ‘successful’ TOC operation (i.e. one where the transfer results in a decision) made at a particular time.

5. A function $P_{continue} : O \times \mathbb{Z} \to (0, 1)$. This function describes the probability that a particular agent will *not* make a decision if control is transfered to it a particular time. For example, $P_{continue}(f, 5) = 0.25$ indicates a 75% chance that agent f will make the decision at time 5.

From these inputs, the agent must find TOC_{opt}, a sequence of TOC operations with maximum expected utility. This formulation differs in several notable ways from those of earlier authors (e.g. [7], [63]), which we discuss in terms of assumptions we have made in the coming sections. The assumptions made serve to simplify the problem for its application to resource allocation, and, as argued below, can typically be relaxed without substantially impacting the results discussed in this chapter.

3.2.1 Assumption of Independence

Previous versions of the TOC strategy generation problem have assumed a valuation function which can be dependent on a preceding sequence of TOC operations, rather than being dependent on the current TOC operation alone. That is, while we assume $V : O \times \mathbb{Z} \rightarrow \mathbb{R}^+$, previous work assumes $V : O \times \{O \cup \emptyset\}^k \times \mathbb{Z} \rightarrow \mathbb{R}^+$, where k is the length of the strategy and a similar assumption is made for $P_{continue}$. In this context, we assume a Markovian property (i.e. a lack of dependency between current and previous TOC operations) not only because doing otherwise can produce a significant increase in the search space and evaluation time, but also because it is not obvious that such dependencies are necessarily realistic or required in certain problem domains. In a fast paced and dynamically changing resource allocation setting, information may lose value very quickly, and indeed, experimental evaluations of the full system, discussed in chapter 5, suggest that this is the norm for the domains of interest.

A consequence of the Markovian assumption is that it can become difficult to model 'partial' transfers of control (PTOCs)[5], wherein the agent asks an entity for preference information rather than actually transferring decision making to the entity. This is because an agent's future decisions will naturally be expected to change if it gains new information from a PTOC operation, violating the Markov property. Although this initially appears to be a substantial loss, we offer both reasons for thinking the loss a minor one, and methods to mitigate this decrease in functionality in situations where PTOCs are still required.

To begin with, we notice that in fact, many PTOC requests are still allowed. The effective constraint is that we allow only PTOCs when the information to be gained will be acted upon immediately or not at all. That is, we allow agents to make PTOC requests which reduce to the form 'Who should I transfer control to next?' or 'May I transfer control to you?' without any trouble. We also allow requests for preference elicitation, (e.g. 'Would you rather I schedule a meeting in the morning or afternoon?'), but only if the information will be used immediately. For example, if answering 'morning' will result in control being transfered to me immediately, but afternoon will not, the previously given question is fine.

As a result, we can abstractly model PTOCs in our system as follows. The target of a PTOC request specifies a subset of the possible entities, which we will call the candidate set. The target also specifies a local estimate of V and $P_{continue}$ for each member of the candidate set. The agent can then asses the optimal subsequent TOC request on the basis of this new information and issue it. To valuate the overall expected utility of issuing a PTOC, we require an estimate of expected utility that would arise from following the recommendation of the targeted entity. This can be obtained either as the average over all possible values the entity might return, or some value estimated through repeated experience. Formally, the value of a PTOC would be expressed as:

$$V_{PTOC}(o, time) = \sum_{x \in R_F} P_{PTOC}(o, x, time) \times V_{FTOC}(x, time)$$

where the function P_{PTOC} is the probability that the PTOC o, if executed at time $time$, will produce an answer indicating that we should make FTOC x next, and V_{FTOC} is simply the valuation function for FTOCs discussed above[1].

The operations themselves are implemented as a combination PTOC/FTOC, and are assumed to occur within a single time step (i.e. Δt time). That is, we assume that PTOCs take much less time to process than FTOCs do. This is not entirely inconsistent with Cheng's vision of PTOCs [5], which always terminated with an FTOC (see figure 3.1).

All that being said, there are good reasons for discounting the loss of arbitrary PTOC support in the first place. PTOC operations require us to have a great deal of knowledge about the entities in our system. We do not need to know merely their ability to make the decision and probability of actually agreeing to make it, but the extent to which they might know the details of other entities' ability to make the decision, and/or probabilistic estimates of how likely the entity is to make a particular response when queried. In the domain of Multiagent Resource Allocation, the entities we deal with will often have limited information about others, and we may not have good methods of estimating the information that they do have, making PTOCs an unreliable tool. Further, unconstrained PTOCs grossly increase the run times of both the new approaches discussed here and of the heuristic search strategies adopted by earlier authors (PTOCs can greatly increase the branching factor of a search) [5].

[1]N.B. This valuation may be incorrect, since the local estimate of V provided by the target of o may differ from the actual function, but it is nonetheless a principled estimation.

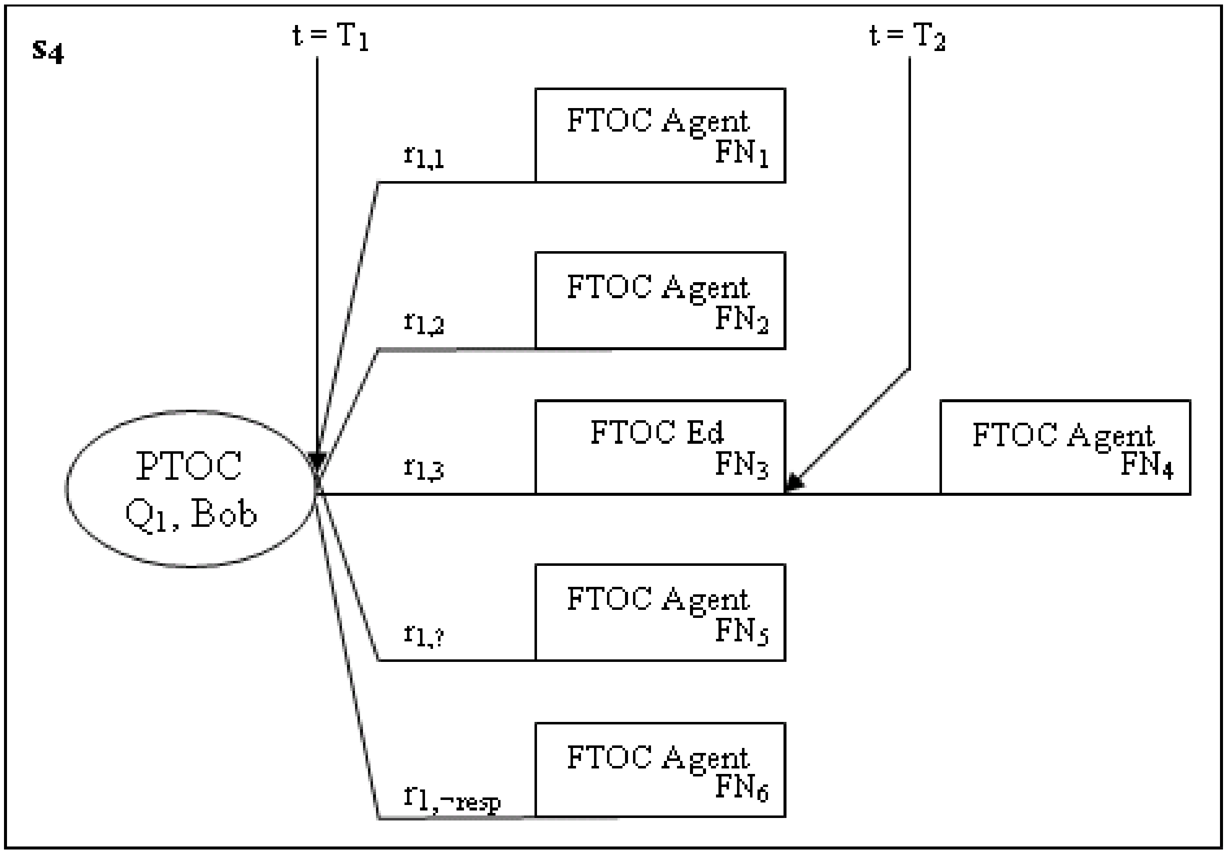


Figure 3.1: An example of a combined PTOC and FTOC where Q1 is "When rescheduling the meeting, which factor should be prioritized?", reproduced from [5]. After the PTOC executed at time T_1, control can be transfered to any of the possible FTOC nodes. Note that '?' indicates a reply of "I don't know" and $\neg resp$ indicates "no response". $T_2 - T_1$ is the time to wait for a response from Ed before the agent proceeds with making the decision itself.

3.2.2 Assumption of Discrete Time Steps

Again, previous authors[63][5] have assumed continuous values for time. That is, their methods did not utilize discrete time steps in the modelling of TOC strategies, but instead assumed that TOC operations could have real-valued lengths. This had an advantage in certain domains, but required use of a valuation function which was continuous and differentiable with respect to time for each resource. In the absence of such a function, these authors resorted to discrete optimization similar to the methods utilized here. Once again, we invoke the application domain as justification for broadening this assumption to the problem as a whole. It is unrealistic to expect continuous and differentiable (and *known*) functions in a rapidly changing environment. For example, the costs associated with issuing a particular TOC operation might have a jump discontinuity with respect to time at the point where the resource is expected to complete its current task, anticipating that the resource will become free for use. Alternatively, an agent's needs might switch suddenly if deadlines are broken. In both of these cases, functions will have non-differentiable points, necessitating the use of discrete optimization. Since these factors are common in resource allocation tasks, we elect to utilize discrete optimization by default.

A more specific example may assist. Suppose that an agent Joe needs 100 widgets made by Monday. If they are delivered by Monday, all is well, and Joe receives a utility of 5. If they are not delivered by Monday, Joe has no use for them (widgets earn utility zero), and in addition, he will need to replace an expensive steam engine as soon as possible. Joe's valuation function for widgets and steam engines is shown in Figure 3.2. These utility functions have numerous points where they are not differentiable, preventing the use of the real valued optimization techniques of prior authors [63][5]

3.2.3 Constraints on TOC Request Frequency

In this model, we assume that TOC operations must be made in each timestep, and that each operation takes the same amount of time to complete. Earlier models, in contrast, assumed that the optimal length of time a TOC operation could take could be estimated (although, as discussed above, in this domain the length of time would be estimated by discretizing the valuation function first), and that there could be periods of time in the strategy where no TOC operation was being conducted (i.e. where the agent was waiting before making another TOC request). We justify these limitations by noting that the latter model can easily be reduced to the former. The use of discrete optimization for finding the length of an optimal TOC operation is equivalent to allowing multiple instances of

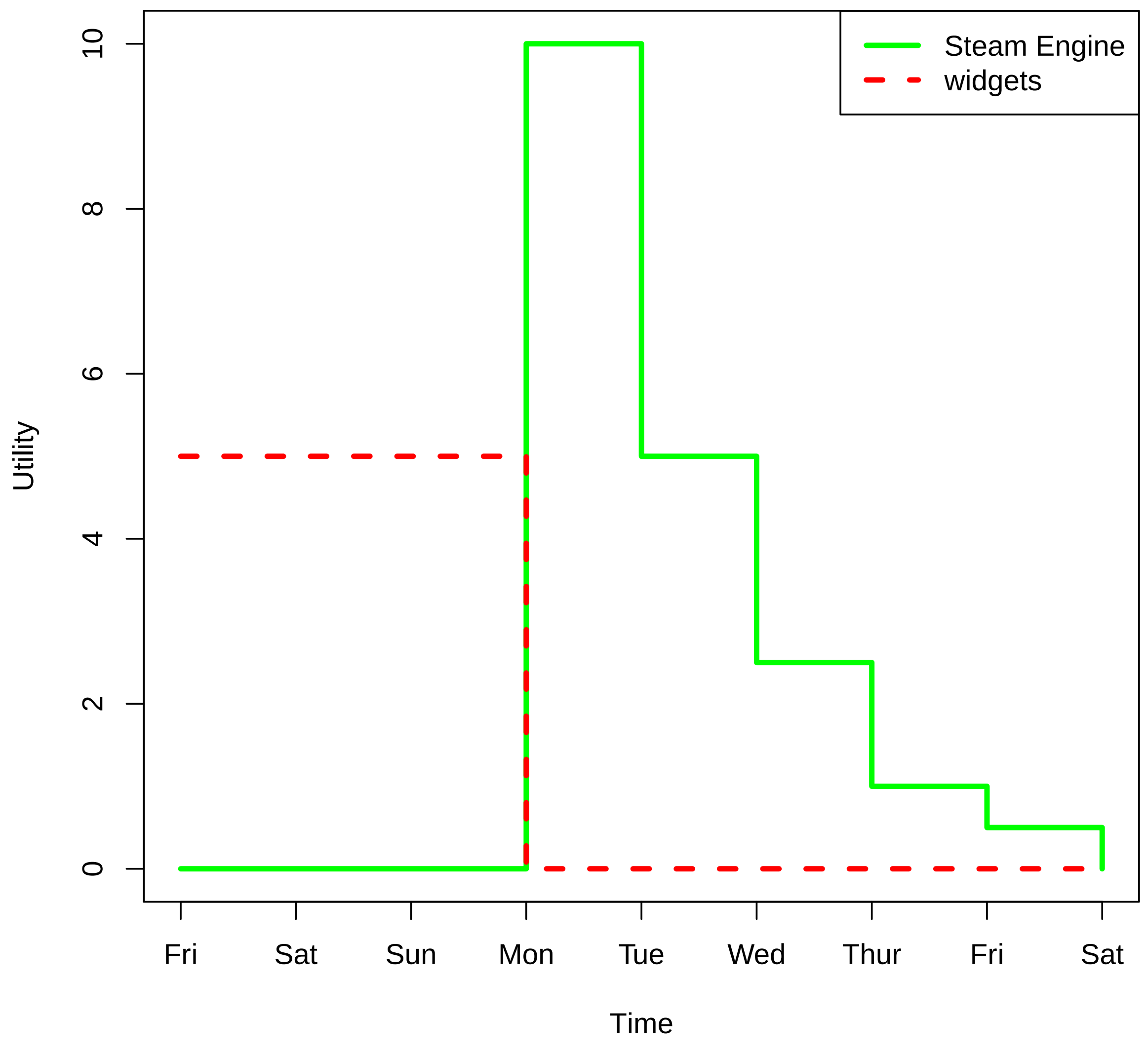


Figure 3.2: A figure showing a valuation function which prevents efficient optimization of TOC strategies using continuous values for time. Notice the jump discontinuities on Monday.

a given TOC operation to appear in the strategy[2], each with a fixed time step length of Δt. The allowance of a 'wait' action is accommodated with the introduction of a dummy 'wait' operation w which has the property that $V(w,t) = 0$ and $P_{continue}(w,t) = 1$ for all $t \in \mathbb{Z}$. Allowing multiple copies of w to appear in a single strategy facilitates the creation of arbitrary length waiting periods.

3.2.4 Example Problem

A concrete example of a TOC strategy generation problem may help to clarify the reader's understanding. Suppose that an agent *Bob* wants to obtain breakfast for *Alice*, who is asleep. Bob can spend at most one hour obtaining breakfast before Alice awakens, and is willing to spend 5 minutes trying any given course of action before changing to any other course. Thus, Δt is 5 minutes, and $k = \frac{60}{5} = 12$. If breakfast is not prepared prior to Alice awakening, Bob generates a utility of 0.

Several actions are possible. Bob can order a fancy breakfast from the *restaurant* down the street, but the restaurant is busy, and might not answer the phone. The restaurant will become more likely to answer the phone as time passes and they work through the breakfast backlog, but the possibility of the delivery arriving before Alice awakens also decreases. This latter value is incorporated into the estimate for V, while the former is found in the estimate for $P_{continue}$. Alternatively, Bob could walk to the greasy spoon down the street, and take out an order there, but such greasy food needs to be eaten relatively soon after it is made, which means ordering it too early might produce a substandard meal. Bob could also make breakfast himself, but his meager cooking skills will yield a sub-par meal. Bob has a belief about the value of each of these alternatives, and the probabilities that they will actually respond to a request to make breakfast. These beliefs comprise the V and $P_{continue}$ functions respectively.

For convenience, we adopt a slightly different notation from that of previous work [5][35][63]. Conventionally, TOC operations are represented as a sequence of alternating times and actions. For example the strategy $\{\{e_1 t_1 e_2 t_2 e_3 t_3\}\}$ would denote transferring control to entity e_1 at time t_1, then, if the decision remains unmade, to e_2 at time t_2 and finally to e_3 at time t_3 if entity e_2 fails to make the decision at all. Since we have assumed a discrete and constant length of time for each TOC operation, and assumed that a TOC operation must be made at each time step, we adopt a slightly more compact notation which omits the times from the strategy, since they are implicit in this representation.

[2]We can still model TOCs which take longer intervals, as long as they are multiples of Δt. This means that we are also resolving how long to wait for a response, for each entity in the TOC.

Thus, in the above example, if $t_1 = 1$, $t_2 = 3$, and $t_3 = 4$, then we would represent the strategy as $\{\{e_1, e_1, e_2, e_3\}\}$, which should be read as "Transfer control to entity e_1 first. Leave entity e_1 in control for two time steps. Then, at time step 3, transfer control to entity 2 for one time step, and then to entity 3 for the final step".

Formally then, the breakfast problem is represented as follows:

$$\{O = \{restaurant, spoon, cook, wait\}, \Delta t = 5, k = 12, V, P_{continue}\}$$

where V and $P_{continue}$ are given in tables 3.1 and 3.2 respectively. To compute the value of a TOC strategy, $S = \{\{s_1, s_2, s_3, s_4, ...s_{max}\}\}$, Bob utilizes the recurrence

$$EU(S,t) = \begin{cases} V(s_1,t) \times (1 - P_{continue}(s_1,t)) + & \\ P_{continue}(s_1,t) \times EU(rest(S), t + \Delta t) & \text{if } S \neq emptylist \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where $rest(S)$ denotes all of S following the first element (i.e. the rest of the list). For example, the strategy $S_{sample} = \{\{wait, wait, wait, wait, restaurant, restaurant, cook\}\}$ represents waiting for the first four time steps, then trying to call the restaurant for two time steps, and finally trying to cook breakfast if the restaurant can't deliver[3]. This strategy has an expected value of 0×1.0 for each of the four waits, since we are certain both to get zero utility and not to make the decision. The first call to the restaurant has value 10, but the restaurant will only answer with probability 0.4. So the value is thus $10 \times 0.4 = 4$. The next call to the restaurant is only made if the first one didn't work out, so it only happens with probability $1 - 0.4 = 0.6$. The call would be worth 9 if it succeeded, but only has a 50% chance of success. Thus the total value is $0.6 \times 0.5 \times 9 = 0.3 \times 9 = 2.7$. The final operation, *cook* is only reached if both calls to the restaurant fail. This happens with probability $(1 - 0.6) \times (1 - 0.5) = 0.3$. The value of cooking is 5, so the expected value of planning to cook at time $t = 35$ minutes is $0.3 \times 5 = 0.15$. The total value for the entire strategy is then: $0 + 0 + 0 + 0 + 10 \times 0.4 \times 0.6 \times (9 \times 0.5 + 0.5 \times 5) = 6.85$.

3.3 Existing Techniques

Several prior works have considered the problem of efficiently generating high quality or optimal TOC strategies. In this section we examine these existing techniques in order of

[3]N.B. we have truncated the remaining 5 timesteps of the strategy, since they are never executed due to the $P_{continue}$ value for cook being zero.

$V(o,t)$	5	10	15	20	25	30	35	40	45	50	55	60
restaurant	10	10	10	10	10	9	7	5	3	1	1	0
spoon	4	4	4	5	5	6	6	4	4	3	3	1
cook	6	5	5	5	5	5	5	3	2	1	0	0
wait	0	0	0	0	0	0	0	0	0	0	0	0

Table 3.1: The valuation function V for the breakfast example. This function represents Bob's beliefs about the quality of obtaining a meal from each of these locations if they agreed to start making it at the corresponding time. For example, Bob believes that if the restaurant starts making his meal at time 5, the meal would have plenty of time to be delivered after it was made, so he could expect to receive the full value of the meal. In contrast, at time 55, even if the restaurant takes Bob's order, it is unlikely they can complete the task before Alice wakes up.

$P_{continue}(0,t)$	5	10	15	20	25	30	35	40	45	50	55	60
restaurant	0.9	0.8	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0	0	0
spoon	0.5	0.6	0.6	0.5	0.4	0.3	0.3	0.2	0.2	0.1	0	0
cook	0	0	0	0	0	0	0	0	0	0	0	0
wait	1	1	1	1	1	1	1	1	1	1	1	1

Table 3.2: The continuation function $P_{continue}$ for the breakfast example. This function represents Bob's beliefs about each location actually agreeing to start making a breakfast order at the corresponding time. For example, Bob believes that the restaurant is very busy at time 5, so it is quite unlikely that they will take his order during that time interval. In contrast, by time 50, the morning rush will be over, so Bob expects to be served immediately at that time.

complexity.

3.3.1 Brute Force

The simplest technique used to generate TOC strategies is the brute force enumeration approach adopted by Jung [35]. In this approach, all possible sequences of TOC strategies are generated one after another. Each strategy is then evaluated using a recurrence similar in form to equation (3.1). The strategy with the highest valuation is then selected for use.

An obvious drawback of this approach is its run time. There are $|O|^k$ possible sequences if no restrictions are used, and $\frac{|O|!}{(|O|-k)!}$ even under the assumption that each TOC operation can be executed at most once appearing in [35]. In spite of this, the approach has valid applications when both the frequency and complexity of TOC strategy generation are expected to be small. Further, a careful enumeration of strategies can limit the space required by the algorithm to $O(|O|)$, which could be advantageous in certain situations, given that more complex approaches tend to scale exponentially in memory as well as run time, especially when constraints are present. This enumerative approach is not considered further in this work, but might be of interest to future practitioners for specific classes of problems. Jung's system also maps more directly onto the formulation of the TOC strategy generation problem adopted in this work, since many of the same assumptions were utilized.

Formally, this method is presented in algorithm 3.1. To efficiently enumerate all possible permutations of O, the algorithm requires an ordering over O, that is, it requires us to specify a binary operator $\prec: O \times O \to \{0, 1\}$ which returns 1 if and only if the first argument precedes the second in the ordering. $\prec$ must be transitive and total. For example, an alphabetical ordering over the names of the elements of O would suffice (provided the names are unique). The algorithm requires this ordering to construct a lexicographic order over the set of possible TOC strategies it could construct from O, and then simply considers the strategies according to their rank in the lexicographic ordering. This avoids the necessity of keeping track of previously considered strategies (since we know we've already considered all the strategies which lexicographically precede the one we're considering at the moment).

A short example may assist in understanding. Suppose that, in the breakfast example above, Bob decides to plan out the first 10 minutes, and to plan the remainder of his strategy at a later point in time. There are 4^2 possible strategies Bob could pick for this interval. We use the valuation function given by the recurrence in equation (3.1). Thus, our call to the algorithm looks like $BruteForceTOC(\{restaurant, spoon, cook, wait\}, k = 2, EU)$. Implicitly, global parameters $\Delta t = 5$, V, and $P_{continue}$ are the functions given in tables 3.1 and 3.2. The algorithm begins on line two by setting *best* to the emptylist, a strategy with expected value 0. This is so the strategy chosen by the agent is certain to be no worse than doing nothing. We assume that the lexicographical ordering required by the algorithm is $restaurant < spoon < cook < wait$. Then the initial strategy generated on line three is $\{\{restaurant, restaurant\}\}$. This strategy has value (following equation (3.2))

$$10 \times 0.1 + 0.9 \times (10 \times 0.2 + 0.8 \times 0)) = 1 + 0.9 \times 2 = 2.8$$

which exceeds the value of 0 for *best*, so the conditional in line 4 is true, and *best* is assigned value $\{\{restaurant, restaurant\}\}$ in line 5. We then enter the main loop. In each iteration of the main loop, the lexicographically subsequent permutation is generated (lines 8-14), and then compared with the current best (i.e. highest utility) strategy (lines 15-17). Once all permutations have been considered, the algorithm terminates and returns the best strategy. Table 3.3 shows the permutations in the order which they are generated, and with the valuations produced for each. A ✓ denotes a permutation which is the best seen so far each time one is encountered. The returned strategy is $\{\{cook, restaurant\}\}$, although any strategy starting with *cook* has the same optimal value, and the *restaurant* TOC operation will never be executed (since $P_{continue}(cook, 5) = 0$).

In testing, we verified that newer (and more complex) algorithms were implemented correctly by comparing their output with that of the simpler brute force algorithm for small input sizes. As expected however, the brute force system cannot scale. Its empirical runtime performance is sketched in Figure 3.3, which shows the expected factorial-like increases in times for a test problem with 5 resources. As in the algorithm, k shows the maximum number of steps in a TOC strategy.

3.3.2 Branch and Bound Search

The seminal work in TOC strategies was Scerri et al.'s electronic elves (E-ELVES) project [62] beginning around the turn of the century. This project aimed to create a semi-autonomous and helpful agent which could take over the burden of mundane cognitive tasks from humans. For example, the E-ELVES agents are often shown scheduling meetings for participants with disjoint schedules. Scerri et al. used a branch and bound search technique [41] to generate strategies efficiently in a single agent domain. Early work on the project closely examined the properties of typical search spaces for TOC strategy generation in this domain. In [63], the issue of efficiently identifying optimal strategies was examined empirically, with the conclusion being that the utilized search algorithm achieved approximately linear scaling in its run time as k increased. Cheng and Cohen [7] extended this strategy generation technique to handle partial transfers of control, including updating

Strategy	Value	best?
$\{\{restaurant, restaurant\}\}$	$10 \times 0.1 + 0.9 \times 0.2 \times 10 = 2.8$	✓
$\{\{restaurant, spoon\}\}$	$1 + 0.9 \times 0.4 \times 4 = 2.44$	
$\{\{restaurant, cook\}\}$	$1 + 0.9 \times 1 \times 5 = 5.5$	✓
$\{\{restaurant, wait\}\}$	$1 + 0.9 \times 0 \times 0 = 1$	
$\{\{spoon, restaurant\}\}$	$4 \times 0.5 + 0.5 \times 0.2 \times 10 = 3$	
$\{\{spoon, spoon\}\}$	$2 + 0.5 \times 0.4 \times 4 = 2.8$	
$\{\{spoon, cook\}\}$	$2 + 0.5 \times 1 \times 5 = 4.5$	
$\{\{spoon, wait\}\}$	$2 + 0.5 \times 0 \times 0 = 2$	
$\{\{cook, restaurant\}\}$	$1 \times 6 + 0 \times 0.2 \times 10 = 6$	✓
$\{\{cook, spoon\}\}$	6	
$\{\{cook, cook\}\}$	6	
$\{\{cook, wait\}\}$	6	
$\{\{wait, restaurant\}\}$	$0 \times 0 + 1 \times 0.2 \times 10 = 2$	
$\{\{wait, spoon\}\}$	$0 + 0.4 \times 4 = 1.6$	
$\{\{wait, cook\}\}$	$0 + 5 = 5$	
$\{\{wait, wait\}\}$	$0 + 0 = 0$	

Table 3.3: A worked example showing the order in which algorithm 3.1 examines possible TOC strategies and the computed values for each full strategy.

Algorithm 3.1 A brute force algorithm for generating TOC strategies, based on [11]. Notice that V_{full} is the valuation function for an entire strategy, for example, the recurrence given in equation (3.2). The algorithm also assumes a lexicographic ordering over O exists, though the ordering can be totally arbitrary. $first(O)$ and $last(O)$ are the lexicographically minimal and maximal elements of O respectively. $x \leftarrow x+1$ sets x to the lexicographically subsequent member of O

```
1:  BruteForceTOC($O$,$k$,$V_{full}$)
2:  $best \leftarrow emptylist$
3:  $current\_strat \leftarrow first(O)^k$ {Starting with the lexicographically first strategy}
    {Evaluate the strategy}
4:  if $V_{full}(current\_strat) > V_{full}(best)$ then
5:    $best \leftarrow current\_strat$
6:  end if
    {Until we've considered the lexicographically last strategy}
7:  while $current\_strat != last(O)^k$ do
      {Advance the current strategy to the next lexicographically larger one}
8:    for $current\_k \leftarrow k$ to 1 do
9:      if $current\_strat[current\_k] != last(O)$ then
10:       $current\_strat[current\_k] \leftarrow current\_strat[current\_k] + 1$
11:       $\forall\ i,\ s.t.,\ k \leq i \leq current\_k :\ current\_strat[i] \leftarrow first(O)$
12:       break
13:     end if
14:   end for
      {Evaluate the new strategy}
15:   if $V_{full}(current\_strat) > V_{full}(best)$ then
16:     $best \leftarrow current\_strat$
17:   end if
18: end while
    {Return the best strategy seen.}
19: return $best$
```

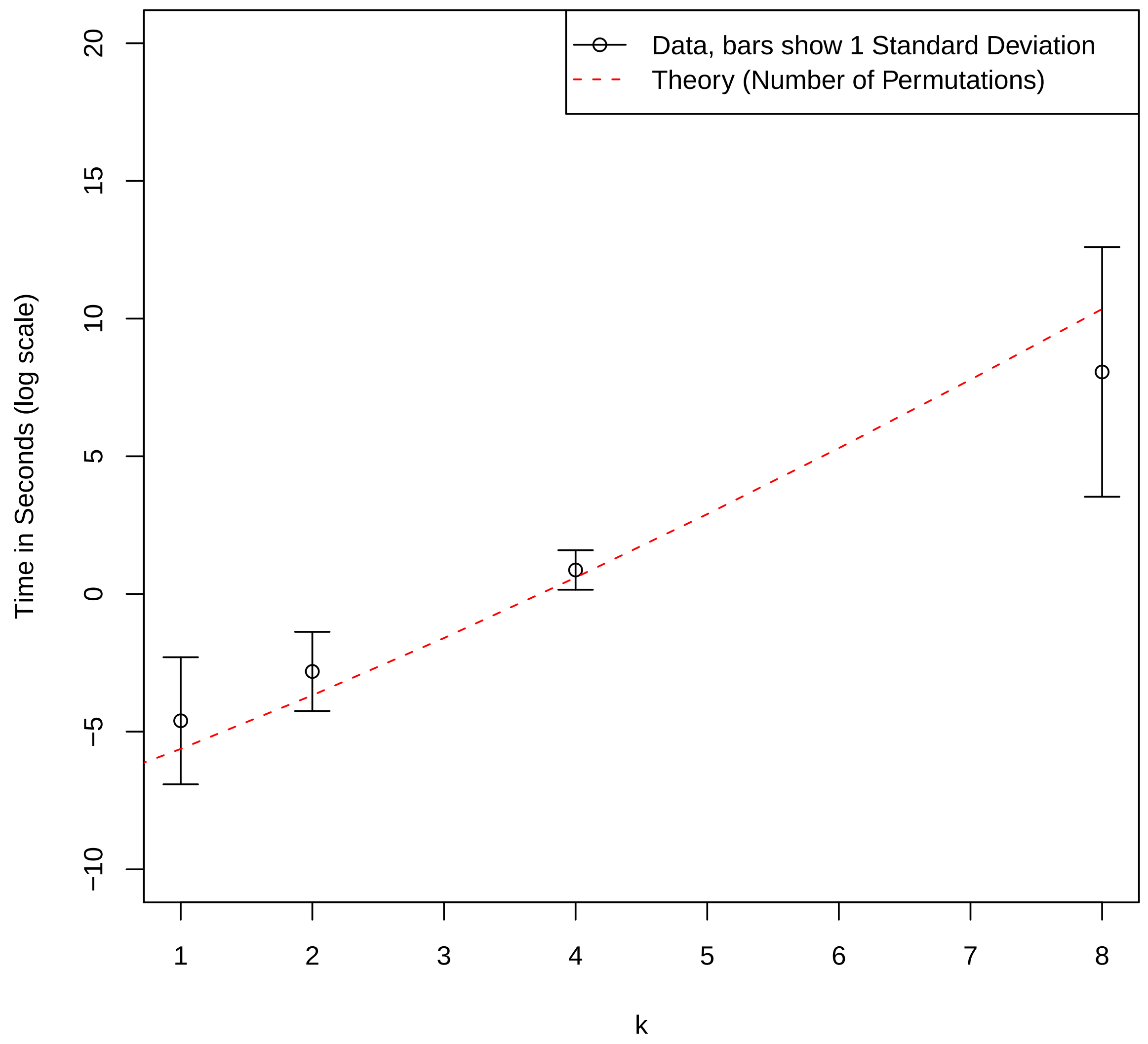


Figure 3.3: A plot showing the observed and predicted run times of a brute force TOC strategy generation algorithm.

the search heuristic to estimate the optimal time to initial PTOCs, observing a similar scaling of computational cost in practice [5]. However, Cheng's method for searching strategies was a simple enumerative heuristic. In this technique, all strategies containing at most one TOC (of arbitrary time length) were generated first. Then all strategies of length 2 were generated. If the improvement between the best strategy with only one TOC and the best strategy with two TOCs was sufficiently small, generation stopped. Otherwise, all strategies containing one more TOC were generated and the same comparison was made, until the change in expected utility was small [5]. A notable drawback of this approach is both that its runtime undergoes a combinatorial explosion if the optimal strategy contains many distinct TOCs, and that it may fail to find the truly optimal strategy if the strategy contains many distinct TOCs and the gain in EU between smaller strategy sizes is small.

Another substantial issue with transferring this algorithm to the resource allocation domain is that the heuristics used to bound the search rely on the existence of smooth or nearly smooth valuation functions. If the functions are differentiable, then it becomes possible to readily compute the optimal time to transition from the current TOC operation to an alternative one. When this optimum is non-existent, the corresponding TOC operation can be removed from the set of viable options, reducing the branching factor of the search. Similarly, if the optimum is global, and yet still below the value of an alternative operation with a maxima at the same point, heuristic pruning may be conducted. This presents a significant problem in domains where functions are empirically defined, as with the tables in the breakfast example, or when they are not smooth and possess many local optima. In resource allocation problems, these possibilities are not unlikely, and indeed, might be expected to appear frequently.

3.4 New Algorithm

In general, TOC strategies are Markovian. That is, the expected quality and probability of an entity providing a decision are not dependent on the entities which preceded it in the strategy. While this is not always the case, the assumption is not entirely unreasonable in the resource allocation domain, especially if care is taken to shield resources from unnecessary requests. For example, the use of proxy agents to filter requests for human entities [10] could allow an agent to query a human resource multiple times without altering the probabilities of response.

3.4.1 Dynamic Programming Solution

If the Markov property holds, then an alternate mechanism for TOC strategy generation appears in the form of a dynamic programming algorithm. The essential idea here is to exploit the Markov property to avoid recomputing optimal suborderings of TOC strategies. For example, in the breakfast example discussed above, Bob can view the problem as a search, in which he chooses first one operation, then the most promising one to follow it, and so on. However, Bob can also view strategy generation as an *arrangement* problem. To accomplish this, Bob simply examines each possible multiset of TOC operations of size k. For each such multiset, Bob determines the optimal order to conduct TOC operations in, assuming he was limited to the TOC strategies in the set, and had to use all the strategies in the set. For example, Bob might draw the multiset $\{wait, wait, cook, cook\}$, and find that the optimal arrangement is $\{\{wait, wait, cook, cook\}\}$ (see table 3.3). However, once Bob has such a multiset, a natural simplification of the problem suggests itself through the recurrence of equation 3.2. Bob can pick cook to start, and then determine the optimal expected arrangement of $\{wait, wait, cook\}$, a subset of the original multiset, starting at time 10 instead of time 5. This allows Bob to fill in a value for the second term in equation 3.2 the generalized multiset $\{wait, wait, cook\}$, rather than for specific arrangements of that set.

For proof of this, consider equation 3.2.

1. Given a multiset $O\times$, let o be an arbitrary element of the set.
2. Suppose that we wish to find the optimal TOC strategy starting with o, comprised of the elements of $O\times$, starting at t.
3. Let $[O\times]$ be an arbitrary ordered list of $O\times$.
4. Let $V(o,t) \times (1 - P_{continue}(o,t)) + P_{continue}(o,t)EU([O\times], t + \Delta T) = c$, for some c.
5. Let $max(O\times)$ be the optimal arrangement of $O\times$, starting at $t + \Delta t$.
6. By definition of optimal, $EU(max(O\times), t + \Delta t) = EU([O\times], t + \Delta t) + c'$ for some non-negative c'.
7. By 5 and 6 above then,

$$V(o,t) \times (1 - P_{continue}(o,t)) + P_{continue}(o,t)EU([O\times], t + \Delta T) + P_{continue}(o,t) \times c' =$$

$$V(o,t) \times (1 - P_{continue}(o,t)) + P_{continue}(o,t)EU(max(O\times), t + \Delta T)$$

8. So, as c' is non-negative, swapping $[O\times]$ for $max(O\times)$ can never decrease the expected value of a TOC strategy under consideration.

Having established the facts above, we can now write a new recurrence in terms of sets describing the expected utility of the optimal TOC strategy.

$$EU_{opt}(O,t,k) = \begin{cases} \arg\max_{o \in O} V(o,t) \times (1 - P_{continue}(o,t)) + \\ P_{continue}(o,t) \times EU_{opt}(O, t+\Delta t, k) & \text{if } t \neq t + \Delta t \times k \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Solving this recurrence is a highly efficient process ($O(n \times k)$ steps), when done backwards in time using dynamic programming. This worst case bound provides compelling reason for its use over heuristic search in practice, even if the search has a better expected case run time. Alternatively, if the Markov property is not entirely existent (as might be the case if, for example, asking an entity changes the probability of their responding at some future time[4]), and we are constrained to issue each TOC operation at most once, then efficiency decreases, but is still worst case exponential ($O(2^n)$), instead of factorial as in the search algorithms and brute force approaches discussed above (these run times are proven in section 3.5 below). Algorithm 3.2 formally specifies the process of finding the optimal TOC strategy when no constraints are present. When strategies are constrained to execute each TOC operation exactly once (for example when the problem domain is inherently non-Markovian), algorithm 3.3 should be used instead, although branch and bound search may be preferable under those conditions (see section 3.5).

An example may clarify the algorithms' workings and the distinctions between the two. Suppose again that, as in the example from table 3.3, Bob needs to compute the optimal TOC strategy starting at time 5 and lasting for 3 timesteps in total. The call to the DPTOC function described in algorithm 3.2 looks like DPTOC($\{restaurant, spoon, cook, wait\}$, 3, V, $P_{continue}$, 5, 5), where V and $P_{continue}$ are as given in tables 3.1 and 3.2. The algorithm starts by setting the value for executing the entire strategy without making a decision to 0, on line 2, and the strategy remaining at that time to the empty list. It then computes the time at the final timestep, which is thereafter referred to as τ, on line 4. In the first pass through the main loop at line 5, the algorithm computes the expected values of executing each TOC strategy at time 5, given that failing to make a decision will produce

[4]As might happen, if humans are bothered with TOC requests they do not wish to answer, their probabilities of responding to (at least immediately) subsequent responses are likely to be lower, violating the Markovian assumption.

an expected value of 0. It begins by computing the expected value of executing $restaurant$ at time $\tau = 15$:

$$V(restaurant, 15) \times (1 - P_{continue}(restaurant, 15) + P_{continue}(restaurant, 15) \times$$

$$EU(emptylist, 20) = 10 \times 0.2 + 0.8 \times 0 = 2$$

Since this new value exceeds the current optimal value, *tmp_best* is set to $restaurant$, and *tmp_value* is set to 2 in lines 11 and 12. The algorithm then repeats this computation for all other TOC operations, giving the values in the first portion of table 3.4 (✓'s denote new optimal values, as before). After considering all $|O|$ TOC operations, the algorithm stores the best option in *best*, the expected utility of that option in $EUBest$, and repeats the process for timestep 10, now under the assumption that failure to reach a decision in timestep 10 will produce an expected utility of 5 instead of 0. This continues until we reach $\tau = 0$, when the final result $\{\{restaurant, restaurant, \mathit{cook}\}\}$ is returned, with value 6.4. The fully computed results are shown in table 3.4.

Algorithm 3.3 behaves in a somewhat similar fashion, but since it assumes that agents can issue each TOC operation at most once, the problem becomes more complex. Now we have to worry about whether a particular TOC operation should be issued at the current time, or some later time. To determine this, we need to consider both the case where this operation is issued now (and thus is followed by some optimal strategy on a set which does not contain it), or later. Again proceeding with the breakfast example above, this algorithm starts by computing optimal strategies for *all* subsets of O of size 1, at time $\tau = 15$, much like algorithm 3.2. Instead of discarding all but the optimal strategy however, all 4 subsets are used as keys in two separate dictionary structures. *plans* stores the optimal TOC strategy for the indexing subset, starting at time 15. *values* similarly stores the value of that optimal strategy. In the next timestep ($\tau = 10$), each of these subsets is *expanded*, which is to say all elements of O which are not already present in the subset are prefixed in turn to the optimal strategy stored in *plans* for the subset of interest. The resulting plans are evaluated, and, if they are the best plans encountered so far containing only the elements of the subset, then they are stored in the *plans* dictionary, and their values in the *values* dictionary. Notice that *multiple* instances of most subsets of size two will be generated. For example, the subsets of size 1 $\{\mathit{cook}\}$ and $\{\mathit{spoon}\}$ will be expanded into strategies (ordered lists) $\{\{\mathit{spoon}, \mathit{cook}\}\}$ and $\{\{\mathit{cook}, \mathit{spoon}\}\}$ respectively, which have the same indexing (unordered) subset $\{\mathit{cook}, \mathit{spoon}\}$. Only one of the two plans (the one with

Strategy	Value	Best in Step	Best After
$\{\{restaurant\}\}$	$10 \times 0.2 + 0 = 2$	✓	
$\{\{spoon\}\}$	$4 \times 0.4 + 0 = 1.6$		
$\{\{cook\}\}$	$5 \times 1 + 0 = 5$	✓	✓
$\{\{wait\}\}$	$0 \times 0 + 0 = 0$		
$\{\{restaurant, cook\}\}$	$10 \times 0.2 + 0.8 \times 5 = 6$	✓	✓
$\{\{spoon\, cook\}\}$	$4 \times 0.4 + 0.6 \times 5 = 4.6$		
$\{\{cook\, cook\}\}$	$5 \times 1 + 0 \times 5 = 5$		
$\{\{wait\, cook\}\}$	$0 \times 0 + 1 \times 5 = 5$		
$\{\{restaurant, restaurant, cook\}\}$	$10 \times 0.1 + 0.9 \times 6 = 6.4$	✓	✓
$\{\{spoon, restaurant, cook\}\}$	$4 \times 0.5 + 0.5 \times 6 = 5$		
$\{\{cook, restaurant, cook\}\}$	$6 \times 1 + 0 \times 6 = 6$		
$\{\{wait, restaurant, cook\}\}$	$0 \times 0 + 1 \times 6 = 6$		

Table 3.4: A worked example showing the application of unconstrained dynamic programming to the breakfast problem. "Best in Step" indicates that the conditional on line 10 was true, while "Best After Step" indicates that *best* was set to this strategy on line 16 after the loop ended.

the higher expected value) is stored at the end of the iteration, which means the number of stored plans of size 2 is the number of subsets of size 2: $\binom{|O|}{2}$. At the next step, the process is repeated by expanding all possible subsets of size 2, and so on. At the final iteration, the optimal plan is returned. A fully worked example for this algorithm is shown in table 3.5. Notice that the local optimum in each step does not appear in the globally optimal strategy. For example, the first round of evaluation finds that ending with $\{\{cook\}\}$ is optimal, but the globally optimal strategy starts with cook instead.

An alternative view of algorithm 3.3 is that of a transformation from a non-Markovian problem to a Markovian one. From a search viewpoint, in the non-Markovian problem, we first pick a TOC operation to try. Now, for our second action, there is some TOC operation we cannot pick, which we can store as a state. As we continue our search, by keeping track of our states, we avoid considering areas of the search space that represent invalid states. From a dynamic programming viewpoint, this process is accomplished through an expansion of the lattice traversed by the dynamic programming algorithm into a high dimensional form. In this case, we can be in one of two possible states with respect to each TOC operation at each lattice point: at the current location in the lattice, we have either already considered the location of a particular TOC operation, or we have not. By creating a lattice point for each possible combination of states, we can simply use a slightly adjusted version of algorithm 3.2 to traverse this much larger lattice efficiently. Algorithm 3.3 amounts to a combination of constructing this larger lattice on the fly (for efficiency purposes), while also traversing it in this fashion.

Algorithm 3.2 A dynamic programming algorithm for finding the optimal TOC strategy when individual TOC requests can be issued multiple times without influencing the value of subsequent requests.

```
1: DPTOC($O$,$k$,$V$, $P_{continue}$, $t$, $\Delta t$):
2: $EUBest \leftarrow 0$
3: $best \leftarrow emptylist$
   {Initialize $\tau$ to the final timestep}
4: $\tau \leftarrow t + k \times \Delta t$
   {Until we have computed the optimal TOC strategy at every timestep}
5: while $\tau \geq t$ do
6:    $tmp\_best \leftarrow best$
7:    $tmp\_value \leftarrow EUBest$
      {Determine the TOC operation which maximizes value at this timestep}
8:    for all $o \in O$ do
9:       $value \leftarrow V(o, \tau) \times (1 - P_{continue}(o, \tau)) + P_{continue}(o, \tau) \times EUBest$
10:      if $value > tmp\_val$ then
11:         $tmp\_best \leftarrow append(o, best)$
12:         $tmp\_value \leftarrow value$
13:      end if
14:   end for
      {Replace the best strategy with the best one for the previous timestep with the best
      TOC operation as a prefix}
15:   $EUBest \leftarrow tmp\_value$
      {Set $\tau$ to the preceding timestep}
16:   $best \leftarrow tmp\_best$
17:   $\tau \leftarrow \tau - \Delta t$
18: end while
    {Return the best strategy seen.}
19: return $best$
```

Strategy	Value	Best Subset	Best After
$\{\{restaurant\}\}$	$10 \times 0.2 + 0 = 2$	✓	
$\{\{spoon\}\}$	$4 \times 0.4 + 0 = 1.6$	✓	
$\{\{cook\}\}$	$5 \times 1 + 0 = 5$	✓	✓
$\{\{wait\}\}$	$0 \times 0 + 0 = 0$	✓	
$\{\{spoon, restaurant\}\}$	$4 \times 0.4 + 0.6 \times 2 = 2.8$		
$\{\{cook, restaurant\}\}$	$5 \times 1 + 0 \times 2 = 5$		
$\{\{wait, restaurant\}\}$	$0 \times 0 + 1 \times 2 = 2$	✓	
$\{\{restaurant, cook\}\}$	$10 \times 0.2 + 0.8 \times 5 = 6$	✓	✓
$\{\{spoon, cook\}\}$	$4 \times 0.4 + 0.6 \times 5 = 4.6$		
$\{\{wait, cook\}\}$	$0 \times 0 + 1 \times 5 = 5$	✓	
$\{\{restaurant, spoon\}\}$	$10 \times 0.2 + 0.8 \times 1.6 = 3.28$	✓	
$\{\{cook, spoon\}\}$	$5 \times 1 + 0 \times 1.6 = 5$	✓	
$\{\{wait, spoon\}\}$	$0 \times 0 + 1 \times 1.6 = 1.6$	✓	
$\{\{restaurant, wait\}\}$	$0.2 \times 10 + 0 = 2$		
$\{\{spoon, wait\}\}$	$4 \times 0.4 + 0 = 1.6$		
$\{\{cook, wait\}\}$	$5 \times 1 + 0 = 5$		
$\{\{spoon, wait, restaurant\}\}$	$4 \times 0.5 + 0.5 \times 2 = 3$		
$\{\{cook, wait, restaurant\}\}$	$6 \times 1 + 0 \times 2 = 6$	✓	✓
$\{\{wait, restaurant, cook\}\}$	$0 \times 0 + 1 \times 6 = 6$		
$\{\{spoon, restaurant, cook\}\}$	$4 \times 0.5 + 0.5 \times 6 = 5$		
$\{\{spoon, wait, cook\}\}$	$4 \times 0.5 + 0.5 \times 5 = 4.5$		
$\{\{restaurant, wait, cook\}\}$	$0.1 \times 10 + 0.9 \times 5 = 5.5$		
$\{\{wait, restaurant, spoon\}\}$	$0 \times 0 + 1 \times 3.28 = 3.28$	✓	
$\{\{cook, restaurant, spoon\}\}$	$6 \times 1 + 0 \times 3.28 = 6$	✓	
$\{\{restaurant, cook, spoon\}\}$	$0.1 \times 10 + 0.9 \times 5 = 5.5$		
$\{\{wait, cook, spoon\}\}$	$0 \times 0 + 1 \times 5 = 5$		
$\{\{restaurant, wait, spoon\}\}$	$0.1 \times 10 + 0.9 \times 1.6 = 2.44$		
$\{\{cook, wait, spoon\}\}$	$6 \times 1 + 0 \times 1.6 = 6$	✓	

Table 3.5: A worked example showing the application of **constrained** dynamic programming to the breakfast problem. “Best for Subset” indicates that the strategy is the optimal arrangement of its components, while “Best After Step” indicates that *best* was set to this strategy after the loop ended.

Algorithm 3.3 A dynamic programming algorithm for finding the optimal TOC strategy when each TOC request can be issued at most once.

```
1: ConstrainedDPTOC($O$,$k$,$V$, $P_{continue}$, $t$, $\Delta t$):
   {We use these variables to remember solutions to optimal subproblems}
2: LET $values$ be a subset/EU map initialized to $\{\emptyset, V(\emptyset, t + \Delta t \times (k+1))\}$.
3: LET $plans$ be a subset/list map initialized to $\{\emptyset, emptylist\}$.
4: LET $best$ be a list/EU pair initialized to $best = \{emptylist, -\infty\}$.
   { We start by computing the expected value of asking each entity *last*, and compute
   *backwards* in time.}
5: for $\tau = t + k \times \Delta t$ to t by $-\Delta t$ do
     { At each time step, we *expand* each previously considered subset in every possible
     fashion. }
6:   for all $subp \in \{x \in keys(values) | length(x) = \tau + \Delta t\}$ do
7:     for all $o \in O$ do

         {Generate a new plan, which is the optimal plan for $subp$ with $o$ prefixed,
         starting at time $\tau$, and valuate it.}
8:       if $o \notin subp$ then
9:         $newplan \leftarrow append(o, plans(subp))$.
10:        $newvalue \leftarrow V(o,\tau) \times (1 - P_{continue}(o,\tau)) + P_{continue}(o,\tau) \times values(subp)$
11:      end if

         {If this combination of TOCs is new or better than any prior arrangement, store
         it as the best strategy for $subp \ \cup \ o$.}
12:      if $o \cup subp \notin keys(values)$ or $newvalues > values(o \cup subp)$ then
13:        $values(t \cup r) \leftarrow newvalue$
14:        $plans(t \cup r) \leftarrow newplan$
           {If it is also the best plan so far, store it.}
15:        if $newvalue > best.value$ then
16:          SET $best.value = newvalue$
17:          SET $best.plan = newplan$
18:        end if
19:      end if
20:    end for
21:  end for
22: end for
23: return $best$
```

3.5 Asymptotic Analysis

This section examines the asymptotic run times of algorithms 3.2 and 3.3 above, proving their worst-case run times, and also contrasting them with search algorithms. The level of efficiency required by the guiding heuristics of search algorithms to outpace the new algorithms is precisely quantified, providing advise for practitioners. Throughout this section we adopt a consistent terminology to allow for concise statements, and to reduce the confusion of O (the set of possible TOC operations) with the Big-Oh notation used for analysis. We denote the cardinality of $|O|$ with n and the number of TOC operations in the optimal strategy with q. Otherwise, terminology is identical to that of section 3.2.

3.5.1 Repeating TOCs

We begin by examining the performance of algorithm 3.2. The algorithm's runtime is in $O(n \times k)$, which is to say, it grows as the product of the number of timesteps that the strategy should have with the number of TOC operations available. Proof is by construction:

1. Algorithm 3.2 does constant work during lines 2-4 and line 19.
2. The loop which begins at line 5 starts at $\tau = t + \Delta t \times k$.
3. At line 17, tau is decreased by Δt during every iteration of the loop which begins at line 5.
4. The loop starting at line 5 runs until $\tau < t$.
5. By 2,3, and 4, the loop starting at line 5 runs $k + 1$ times.
6. During each iteration of the loop starting at line 5, constant work is done for lines 6-7, and lines 15-17.
7. The for loop starting at line 8 and running to line 13 runs n times (once for each element of O).
8. During each iteration of the for loop starting at line 8, constant work is done.
9. By 6, 7 and 8, the work done between lines 8 and 13 is in $O(n)$ for each iteration of the loop starting at line 5.

10. By 1, 5 and 9, the total work done in algorithm 3.2 is in $O(n) \times O(k)$, which is in $O(n \times k)$.

$\square$

3.5.2 Non-repeating TOCs

Asymptotic analysis for algorithm 3.3 is somewhat more involved. The essential idea is that of a sum over subsets of O. Series identities throughout were obtained using Wolfram Alpha [43], an online database which accepts natural language queries.

The runtime for this algorithm is on the order of

$$R(n,k) \in O\left[n \sum_{m=1}^{k} \binom{n}{m-1}\right] \tag{3.3}$$

Proof is again by construction:

1. The work done from the start of algorithm 3.3 to line 4 is constant.
2. The loop starting at line five (the 'main loop') runs in steps of $-\Delta t$.
3. The main loop starts with $\tau = t + k\Delta t$.
4. The main loop runs until $\tau \leq t$.
5. So, the main loop runs a total of k iterations, by 2,3 and 4.
6. The subloop on line 6 runs once for each item added to *values* in the last iteration of the main loop.
7. During the m^{th} iteration of the main loop, *values* has keys for all subsets of O with cardinality of size m added to it.
8. By 6 and 7, the subloop on line 6 runs once for each subset of size $m-1$, which is just $\binom{n}{m-1}$.
9. The subloop on line 7 runs n iterations, once for each element of O.
10. The body of the subloop on line 7 contains only constant time operations.

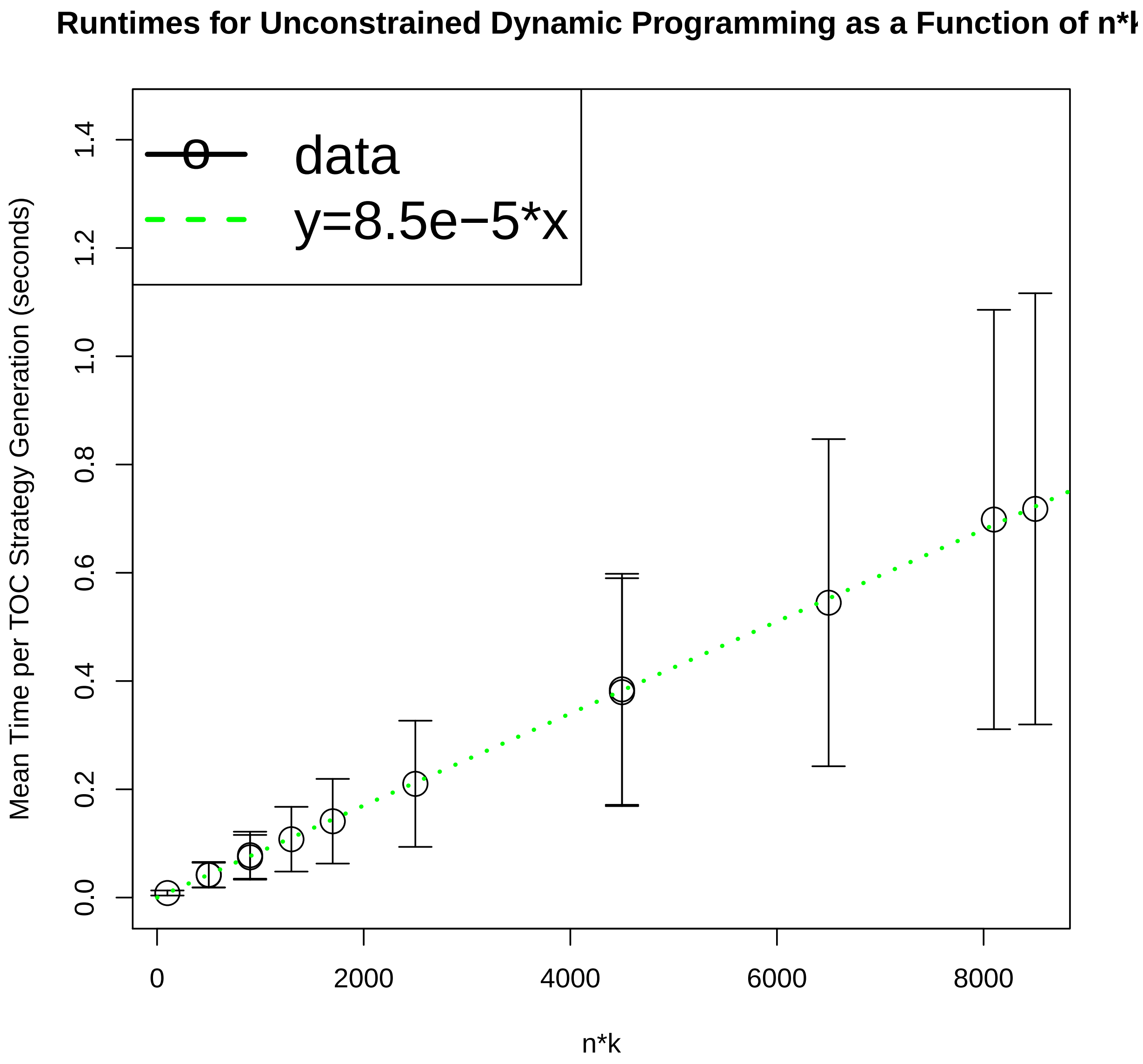


Figure 3.4: A plot showing both the theoretical run time and mean simulation runtimes for algorithm 3.2.

11. By 9 and 10, the subloop on line 7 costs $O(n)$ work.
12. The subloop on line 6 contains only the subloop on line 7.
13. By 8 and 12, the subloop on line 7 costs $O(n\binom{n}{m-1})$ work.
14. By 13 and 5, the main loop costs $\sum_{m=1}^{k} O(n\binom{n}{m-1})$ work, which is equivalent to $R(n,k)$.

□

As an alternate and less formal proof, note that, as in algorithm 3.2, we do $O(n)$ work at each lattice node. Since there is one lattice node for every possible combination of states, and the summation in the runtime bound simply counts the number of combinations of states, the run time is $O(n\left[n\sum_{m=1}^{k}\binom{n}{m-1}\right]$.

Equation 3.3 has a closed form representation. The closed form is given by Wolfram Alpha [43] as

$$n\sum_{m=1}^{k}\binom{n}{m-1} = n \times (2^n - \binom{n}{k}\,{}_2F_1(1, k-n; k+1; -1)) \tag{3.4}$$

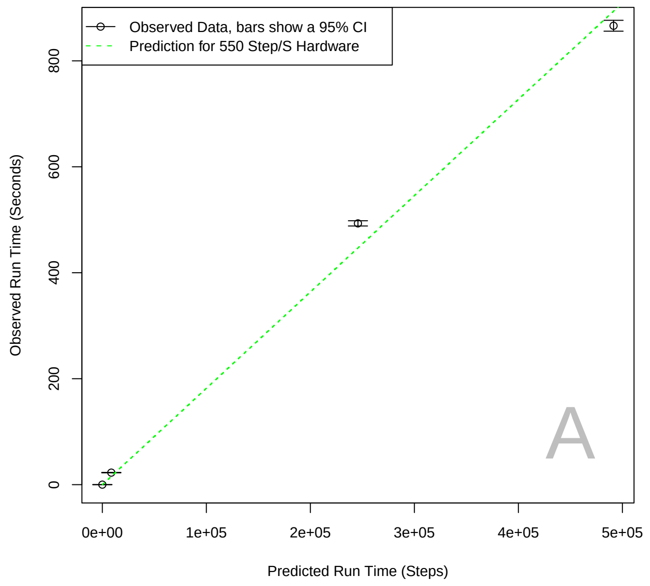
Where ${}_2F_1$ is the hypergeometric function

$${}_2F_1(a,b;c;z) = \sum_{n=0}^{\infty}\frac{(a)_n(b)_n}{(c)_n}\frac{(z)_n}{n!}$$

Although interesting, this information is of limited value, as, for practical values of n and k, the hypergeometric function is not significantly easier to compute than the series representation for the runtime of algorithm 3.3.

Figure 3.5 shows the observed and predicted runtimes for algorithm 3.3 at different values of n and k. We observed a small effect from increasing n which is not immediately explained by our predictive model, however, this appears to be the result of an inefficient indexing procedure (linear in n), and we conclude that the implementation of the algorithm is correct, and that the observed run times are consistent with those we predicted. The large constants are likely due in part to the high memory usage of the algorithm.

An interesting item to note about the above worst case run time is that, for many practical problems, our solution will be substantially faster than 2^n, because long plans



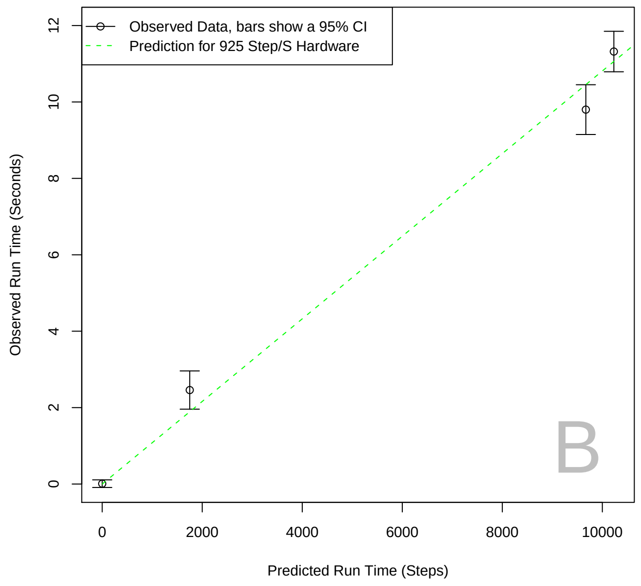


Figure 3.5: Graphs showing the predicted and observed runtimes for algorithm 3.3. Chart (A) shows the runtimes for $n = 15$ with values of $k \in \{1, 4, 8, 16\}$. Chart (B) shows the same values for $n = 10$.

are unlikely to be useful if circumstances are changing rapidly. For example, if we are in a hospital with 200 medical resources, but where things are changing fast enough that there is very little use in planning more than 5 timesteps ahead, then, computing equation 3.4 with values of 200 for n and 5 for k, we find that we need to examine only around 2^{37} possible states. In contrast, if both values are set to 200, 2^{200} states would have to be examined instead.

3.5.3 Comparison with Search, and Minimum Search Efficiencies

Given the results above, we expect that, when search is unconstrained, use of the presented dynamic programming algorithm will always be preferred, since its costs grow linearly in the length of the strategy desired. While branch and bound search is observed to be highly efficient in practice, and in fact, has a faster average run time than the new unconstrained algorithm [63], its runtime is uncertain. This uncertainty may not be worth the price of a marginal increase in runtime, especially in medical applications.

In the constrained case, the branching factor of search decreases with depth, leading to a worst case runtime of $n!$ rather than the n^n of the unconstrained case. Here the advantage of the new algorithm is less clear. While the runtime uncertainty accompanying usage of heuristic search may still be an undesirable factor, the best case exponential-like runtime of the new algorithm is similarly undesirable. Nevertheless, we expect that there are situations where the new algorithm might be preferred to the old, and so we aim to provide a quantification of those circumstances for use by practitioners.

We examine the log ratio of runtimes to quantify the 'minimum heuristic efficiency' required for a heuristic guided branch and bound search to outperform the new constrained algorithm. That is:

$$MHE = log_2(\sum_{m=1}^{k} \binom{n}{m-1} \times \frac{n}{n!}) = log_2(\sum_{m=1}^{k} \frac{n}{(n-m+1)!(m-1)!}) \tag{3.5}$$

In this equation, Positive values of MHE denote situations where search is certain to perform better. A positive value of 1 indicates that the new algorithm will take twice (2^1) as long as the brute force algorithm, which is the worst that any search heuristic could perform, while a value of 2 indicates that the new algorithm will take four (2^2) times as long. Similarly, negative values indicate that the new algorithm examines at most 2^{-x} of the search space, so that any search heuristic must examine, on average, a smaller portion of the search space than 2^{-x}. Figure 3.6 depicts a surface plot for MHE for all legal values of k with up to 100 resources. Although hard to see in this plot, there is a small cluster of points with values above 0, indicating that the old algorithms are certain to be preferred. Figure 3.7 shows an exponentiated version of MHE focused on this region. In this plot, values are linear, so that a value of 3 indicates that a search algorithm would perform at least 3 times as well, while a value of 0.5 indicates that the search algorithm would perform at least half as well. In general, it seems that for values of $n > 8$, or $k > 8$, the new algorithm is to be preferred unless very efficient heuristics (i.e. bounding heuristics which discard a very large percentage of the search space) are available. Figure 3.6 can give practitioners a better idea of how efficient their heuristics should be in order to prefer search. For example, if a heuristic needs to examine only one millionth of the search space (roughly 2^{-20}), then that heuristic is to be preferred in situations where $MHE > -20$, corresponding roughly to the region where $n < 10$ and $k < 10$.

3.6 Conclusion

To sum up, in this chapter we have discussed in detail the problem of finding optimal TOC strategies with a view towards using them in resource allocation problems. We have outlined a number of assumptions which we argue are reasonable in the resource allocation domain and which simplify the problem, including the use of discrete time steps, Markovian independence for TOC operations, and the requirement that TOC operations be performed in all time steps. We then present two new algorithms which use dynamic programming to generate optimal TOC strategies under a fully Markovian problem space, and under one where repeated interactions with the same entity can be non-Markovian. We provide a detailed analysis of the efficiency of the new algorithms, and conclude that they should be preferred to existing techniques in a large number of problem domains, with a concrete performance metric which practitioners can use to determine whether the new algorithm can offer efficiency gains in their problem domain. Further, we note that the application of dynamic programming to this problem has not been attempted prior to our work, making this a novel contribution as well as a useful one. Although we set out to find optimal arrangements of TOC operations, rather than finding the optimal issuing times and lengths of TOC operations as other authors have, our system is still capable of solving these problems in many domains, in a more efficient fashion.

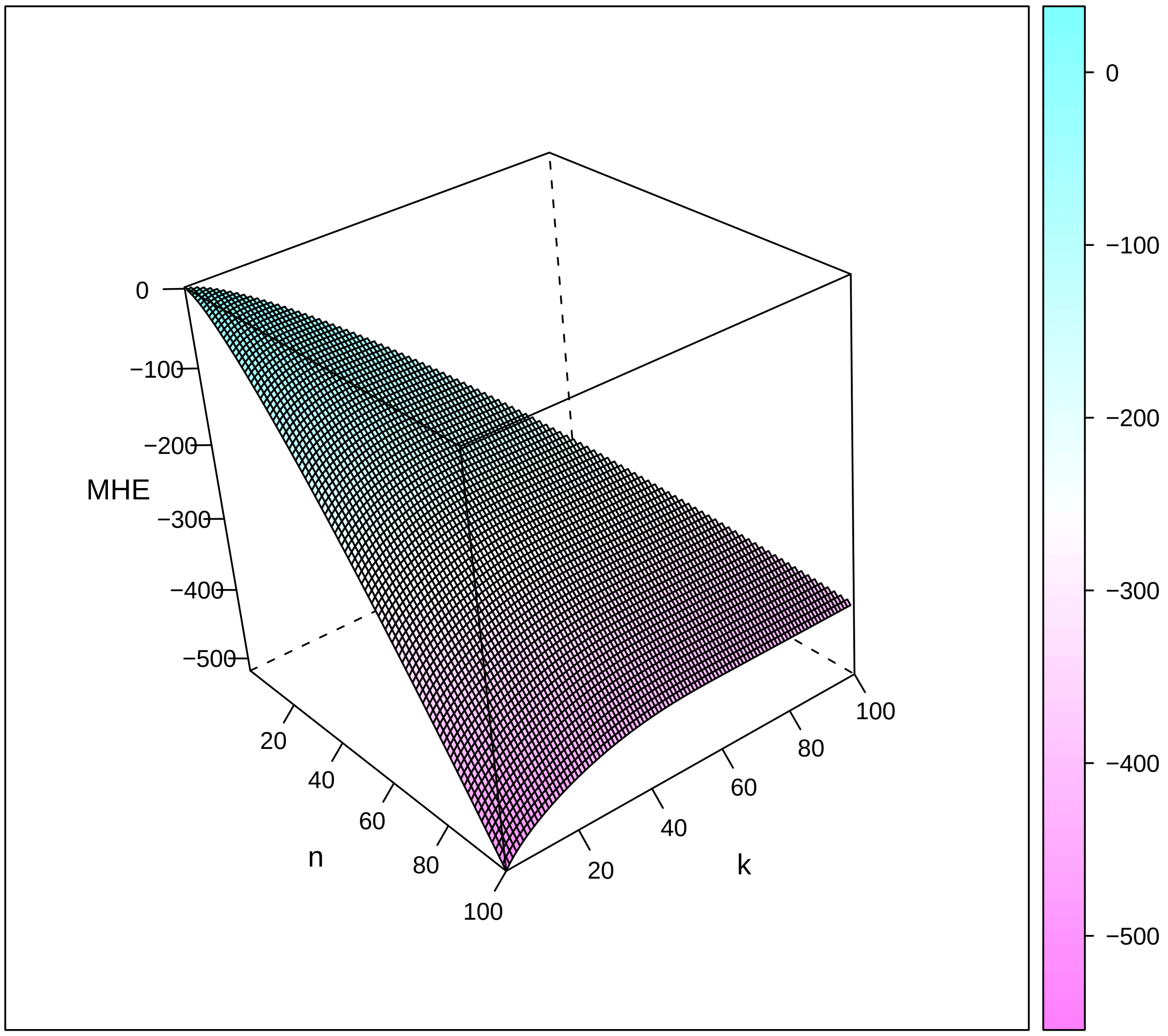


Figure 3.6: A surface plot of MHE (equation 3.5). More negative values indicate exponentially greater advantages for using the new algorithm.

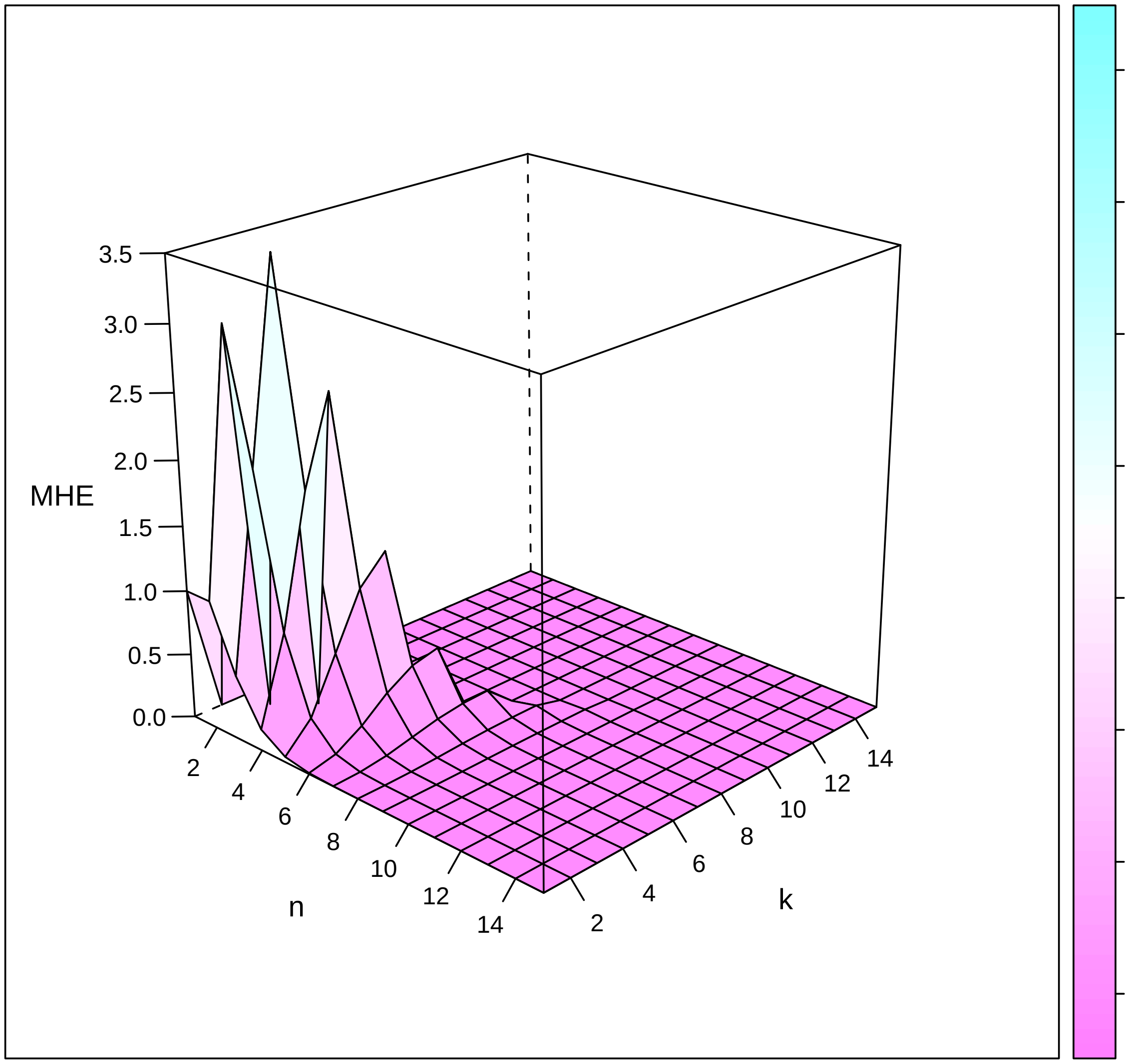


Figure 3.7: An exponentiated surface plot of MHE (equation 3.5). Values greater than 1 indicate places where the new algorithm is certain to perform poorly.

Chapter 4

Estimating Preemption Costs with TOC Strategies

This chapter deals with one of the primary insights of the thesis, namely that transfer of control strategies can be used to provide a reasonably accurate estimate of the opportunity costs associated with holding a particular resource. This fact informs the construction of a fully distributed system for resource allocation that heuristically maximizes utilitarian social welfare, without using money, while still allowing for (partially) self-interested agents.

The material is presented in several blocks. First, we define the distributed resource allocation problem abstractly. We then motivate the need for a more accurate estimate of the opportunity costs associated with holding a resource through the idea of a "preemption cycle", and the philosophical reasons for preemption cycles being an undesirable property of distributed resource allocation systems. Finally, we demonstrate how preemption cycles can be resolved using the idea of a "contingency plan", and how TOC strategies can form the basis of such plans.

4.1 Problem Statement

In general, resource allocation problems under consideration in this work consist of a 6-tuple $\{A, R, T, C, PT, \Pi\}$[1], comprised of:

[1]Note, in chapter 5 we actually define problems as a 7 tuple, but that problem description can be converted to this one by folding the bother cost modelling into the expected utility function to produce a cost function.

- A set of Agents A.
- A set of Resources R.
- A set of Tasks T.
- A cost function

  $$C : \mathbb{N} \times \mathbb{Z} \times T \times \mathbb{Z} \to \mathbb{R}$$

  which describes the expected change in utility associated with finishing a specified task ($\in T$), which has waited a given amount of time ($\in \mathbb{Z}$), at a new time ($\in \mathbb{Z}$), instead of at an old time ($\in \mathbb{Z}$).
- A processing time estimation function

  $$PT : T \times R \to \mathbb{N}$$

  which describes the number of timesteps taken for a particular resource to process a particular task.
- A transition function

  $$\Pi : T \times \{T \cup \emptyset\} \times R \to \{r \in \mathbb{R} | 0 \leq r \leq 1\}$$

  which describes the probability that, if a resource is asked to switch to processing a new task (i.e. at the receiving end of an FTOC), from a (possibly null) old task, the resource will agree to make the switch. More generally, Π is the transition *policy* for the system, describing when and how local acquisitions of resources can take place.

At the beginning of a resource allocation problem, each agent is assigned a task. At each timestep, each agent can ask a single resource to begin processing its task. The resource accepts or declines the switch based on the probability provided by Π. Additionally, if any task t has been processed by a single resource r for $PT(t, r)$ timesteps, the task is declared complete. The agent owning the task is reassigned to a task which currently has no agent (if any), and the resource is now free for use by other tasks.

The objective of the task is to complete processing of all tasks in T in an order which minimizes the total cost:

$$\sum_{t \in T} C(t_{completion}, 0, t, 0)$$

where $t_{completion}$ is the time at which task t was completed.

4.1.1 Example Problem Formulation

For example, consider a resource allocation problem which takes place in a woodworking shop. There are two resources, the *handsaw* and the *table saw*. Two agents, *Bob* and *Margaret* work in the shop, and have 2 tasks, *Cut*_2×4, *make_puzzle*.

Suppose that cutting a

$$2 \times 4$$

takes two timesteps with the handsaw, and one timestep with the table saw. Each timestep it goes unfinished for incurs a cost of one unit. Similarly, suppose that making the puzzle takes 8 timesteps with the handsaw, but only 3 steps with the table saw, and incurs a cost of 4 units for each timestep that it goes unfinished.

Finally, suppose that the workshop's policy is to allow anyone to take a tool which is not in use, and to take a tool which is in use if the costs associated with taking the tool exceed the costs associated with not taking it.

Formally then, we can describe this problem as:

- $A = \{Bob, Margaret\}$
- $R = \{handsaw, table\_saw\}$
- $T = \{Cut\_2 \times 4, make\_puzzle\}$
- $C(new, current, t, wait) =: new - current$ if $t == Cut\_2 \times 4$ and $(new - current) \times 4$ otherwise.
- $PT(t, r)$ is given by[2]:

t/r	*handsaw*	*table_saw*
*Cut*_2×4	1	2
make_puzzle	3	8

- and $\Pi(new, cureent, r)$ 1 if switching costs less, 0 if it costs more[3].

Suppose that, at the initial timestep, Bob is assigned the $Cut\_2 \times 4$ task, and $Margaret$ is assigned the *make_puzzle* task[4]. Both Bob and Margaret will prefer to have the table

[2]Note: Agents can all view the entire PT function, but do not know what other jobs may be present in the system.

[3]This definition has been deliberately left vague. A more rigorous definition of the relative costs appears later in the text.

[4]Algorithms for initial assignments are discussed later in the text.

saw, since it allows them to perform their work faster. If Margaret takes the saw first, she will not give it to Bob (since it costs more to delay her task than it costs to delay Bob's). Bob wastes time asking for the tablesaw, and then settles for the handsaw. Thus, the distributed allocation results in Margaret being assigned the table saw during timesteps 1, 2 and 3 (after which her task is complete), and Bob being assigned the handsaw during timesteps 2 and 3 (after which his task is likewise finished). Note that the optimal allocation is to assign the table saw to Margaret at times 1, 2 and 3, and the handsaw to *Bob* at times 1 and 2, so some efficiency is lost.

4.1.2 Philosophy of Preemption Cost Estimation

This section provides a philosophical motivation for allowing transition policies which include the potential for resource *preemption*, that is, policies which allow agents to take resources from owners who may not wish to relinquish them.

A simple, preemption free, resource allocation policy is to have agents behave as rational, self-interested, utility maximizing creatures. This approach makes distributing resource allocation among agents simple because agents do not have to worry about what other agents think. All they need to do is determine how much a resource is worth to them, and hold on to useful resources unless they are offered something they prefer in trade. Although algorithmically straightforward, this policy has great potential to produce sub-optimal allocations. For example, an agent with great need, but nothing to trade, may be left without a resource indefinitely, while a lucky agent with only minor needs, but many resources to trade, may acquire enormous benefit. For a real world example, consider medical treatment. In a for-profit medical system, agents (acting on behalf of patients) which have more monetary means can obtain high quality care (that is, more skilled doctors) for minor ailments, even as those without such means avoid medical treatment for serious conditions because of prohibitive costs. The allocations produced by this system are inefficient, but have the advantage of being non-coercive: no agent is forced to relinquish (against their will) resources which they possess.

In this work, we adopt the philosophy that non-coercion is, in many resource allocation problems, desirable but ultimately subordinate to efficiency when the collective preferences of the user base are taken into consideration. To justify this position, we adopt an argument based on John Rawls' famed "veil of ignorance" justification for a bill of rights which contained coercive resource transfers [56]. The argument proceeds thus: Assume that potential users of the system are rational, self-interested beings, but that they do not know any of their individual wants or needs. In the language of game theory, we would assume that the agents do not know their own types (i.e. that they are deciding whether to use the system ex-ante). Assume that the users of the system *do* know the distribution of wants and needs for users of the system (the distribution of types). Then, Rawls argues, the agents will be willing to pick a policy for allowing coercive resource swaps ex-ante, which allows users of the system with great need to take things from those with lesser needs. This behavior is rational because users (especially human users [73]) will accept the promise of small costs if they receive an advantageous type in order to hedge against the possibility of very high losses if they receive a disadvantageous type.

An example may assist. Suppose that potential users of a hospital are asked to design the policy for determining who will receive treatment. The users do not know a priori whether they will get cancer, or be hit by a car, or live happily until old age, but they do know the distribution of those events, and they know how vital it is that people with these conditions receive immediate treatment. With this information, it is rational for each agent to agree to a policy which prioritizes those in need of urgent treatment at the expense of those with less urgent conditions, in a fashion which may be coercive. For example, a patient scheduled for cosmetic surgery may have their scheduled timeslot preempted by a patient with a life threatening condition, even if doing so is clearly not in the best interest of the cosmetic patient, who might have scheduled say, 3 vacation days around the surgery's timing. Further, *human* users exhibit a tendency to overestimate the probability of comparatively rare events (like being struck by a car), and so, we argue, are even more likely to agree to such an allocation policy [73].

An alternative approach to addressing the sub-optimal allocations arising from self-interested systems is the incorporation of a form of currency, which can be assigned to agents proportionate to their needs. Self interested agents can then be convinced to relinquish their resources in exchange for monetary compensation, and urgent patients can still obtain treatment. Paulussen et al [50] utilize one such 'market mechanism' in a medical resource allocation system called MediPage, for example, with some good success.

A drawback of these approaches is the potential for agents who are merely lucky, and not truly needy, to acquire a large amount of currency through rent-seeking behaviors. Suppose for example, that an especially fortunate agent enters the system with a small amount of currency at a time when the system is mostly idle. Such an agent can claim the most valuable resources at low costs. When new agents with more urgent conditions enter the system, they must pay this speculating agent to acquire the resources it has claimed, causing the agent to accumulate a large amount of currency and thus allowing it to behave as though it were an agent with a high priority task (since it can exchange this money for higher quality care). Although various mechanisms can be imposed on agents to prevent this sort of abuse, it is not clear that such mechanisms are always rational, and thus, agents may not be willing to play by the imposed rules.

For these reasons, we argue that, in many tasks, an ideal distributed resource allocation system should be free of money, and should allow for coercive (preemptive) transfers of resources in cases of urgent need.

4.1.3 The Costs of Preemption

In the problem described above, Π takes the role of an allocation policy, that is, a set of rules describing (perhaps probabilistically) how two agents should locally resolve a conflict over a particular resource. As described philosophically above, we would like to allow preemptions to occur when the preempting agent has a greater need of the targeted resource than the preempted agent. To estimate this in a principled fashion, we would like to be able to estimate four quantities. Suppose that agent a_1 wants to preempt the resource r presently being used by agent a_2. Then the quantities we want to estimate are:

- $EU_{a_1,r}$, the expected utility of having a_1 take ownership of r.
- $EU_{a_1,\bar{r}}$, the expected utility of not letting a_1 take ownership of r (that is, an estimate of what will happen to a_1 if treatment is not allowed).
- $EU_{a_2,r}$, the expected utility of having a_2 maintain ownership of r.
- $EU_{a_2,\bar{r}}$, the expected utility of taking r from a_2 (that is, an estimate of the consequences for a_2 of allowing the preemption).

If we know these four quantities, then the expected utility of transferring r from a_2 to a_1 is given by

$$EU_{Preempt}(a_1, a_2, r) = (EU_{a_1,r} + EU_{a_2,\bar{r}}) - (EU_{a_2,r} + EU_{a_1,\bar{r}}) \tag{4.1}$$

or, the expected benefits of allowing the preemption minus the opportunity cost of denying it, and letting a_2 keep the resource instead[5]. It is important to note that the opportunity costs must be considered because a true need cannot occur when there are other resources of comparable value which are idle and which the preempting agent could simply claim as its own. Similarly, if the preempted agent can claim other resources to solve its task, the real costs of forcing it to give up its resources will be lower.

[5]Note: We assume that agents have some common scale for expressing the value of resources, although we consider other alternatives in section 8.2.

As we shall see, estimating the value of *not* possessing r is a non-trivial task, which can prevent us from accurately reasoning about the costs of a preemption.

A trivial approach to estimating the value of the term $EU_{a_2,\bar{r}}$ from equation 4.1 would be to compute the costs of having a_2 yield up r, wait until a_1 is done with it, and then take r back again. That is, assume:

$$EU_{a_2,\bar{r}} = C(PT(t_2, r) + PT(t_2, r), comp(t_2), t_2, wait(t_2))$$

and

$$EU_{a_1,\bar{r}} = C(PT(t_1, r) + comp(t_2), comp(t_1), t_1, wait(t_1))$$

where $comp(t_x)$ is the time that task x would complete in the present allocation, $wait(t_x)$ is the time that task x has already spent waiting to be processed, and C is the problem specific cost function.

This appears to be a reasonable estimate, but it has two problems. First, this method clearly has the potential to overestimate costs. Take the workshop example discussed earlier in this chapter. Suppose Bob has taken the table saw, and Margaret wants to use it. When Bob (a_2) estimates the costs associated with giving up the table saw, he will consider the cost of waiting until Margaret is finished, and then using the table saw. However, Bob can of course use the handsaw instead, and indeed, can finish his task with the handsaw before the table saw will become free. Consequently, if Bob uses this method he will overestimate the costs of relinquishing the table saw.

Further, this method can actually underestimate costs by failing to consider the desires of other agents. For example, suppose we add a third agent Charlie, who has a task that costs 3 per timestep to delay, and that can only be processed using the table saw. Using this method of estimating preemption costs, Bob assumes that he will get the table saw back after Margaret is done with it, but in fact, the policy will allow Charlie to claim it for his task, which is more urgent than Bob's, but less urgent than Margaret's. Consequently, if the table saw really were the only resource, Bob's estimate of the preemption cost could be arbitrarily bad, assuming Charlie's job can take arbitrarily large amounts of time.

Because of these deficiencies, we would like to use a method of preemption cost estimation which incorporates both the potential for a_2 to acquire resources other than r while it waits, and the potential for agents other than a_2 to seize resources a_2 desires.

4.2 Preemption Cycles

A significant problem which arises in the type of resource allocation problem described in section 4.1 is a form of deadlock which we will refer to as a "preemption cycle". One of the major contributions in this work is a method of resolving preemption cycles by using TOC strategies to provide a more accurate estimate of the opportunity costs associated with a resource preemption.

4.2.1 What is a Preemption Cycle?

A preemption cycle is a cyclic dependency that arises when estimating the costs of *not* owning a resource. For example, suppose we adopt the following procedure for estimating $EU_{a_2,\bar{r}}$:

1. Iterate over the set of resources excluding r.
2. For each resource, compute the probability that a_2 would succeed in acquiring the resource, and the expected value of acquiring it.
3. The expected value of a_2 giving up r is the maximum expected value of the resources a_2 could acquire after giving up r.

Unfortunately, as an attentive reader may have already noticed, computing the probability of a_2 acquiring the resource is a call to our policy Π. Since Π is supposed to depend on the relative need of the two agents for the resource, this will require estimating $EU_{Preempt}$ for some other agent/resource combination. Similarly, accurately estimating this quantity requires computing further preemption costs *ad nauseam*. Naturally, computing indefinitely deep preemption estimates is infeasible, and so we must find some other method of resolving the cycle.

An example will facilitate understanding.

4.2.2 Example

Suppose we have three agents, Bob, Margaret, and Charlie, who are solving an allocation problem with the resources saw, $knife$, and $file$. Bob needs to cut a board. He can do this easily with a saw (2 steps), or with difficulty using the knife (3 steps). The file is not useful for this task. Margaret needs to engrave a chair. This can be done quickly with the knife (2 steps), or more slowly with the file (3 steps). The saw is not useful for Margaret's task. Charlie needs to cut a metal pipe. With the saw this will take 2 steps, while with the file it will take 20 steps. The knife is not useful to Charlie. Suppose our initial allocation is such that Bob has the saw, Margaret has the knife, and Charlie holds the file. Under this allocation, Charlie will take 20 steps to finish his task, which is clearly undesirable.

To fix this situation, Charlie attempts to preempt the saw, presently in use by Bob. This triggers a sequence of events to compute the cost of the preemption:

1. Charlie asks Bob for the saw.
2. Bob doesn't want Charlie's resource (the file), which is useless to him. Since this would increase the expected time for Bob to complete his task to an indefinite quantity, Bob cannot allow a direct swap.
3. In order for Bob to finish his task without the saw, Bob must have the knife, so Bob ask Margaret whether he can preempt the knife.
4. Much like before, Margaret cannot make a direct swap with Bob, both because Bob is already trying to give his resource away, and because Margaret cannot use the saw in any case.
5. Margaret now asks Charlie for the other tool she *can* use, the file. However, Charlie will only relinquish the file if he can obtain the saw.
6. Charlie asks Bob for the saw, beginning another iteration of the cycle.

This sequence is depicted visually in figure 4.1.

4.2.3 Possible Solutions

There are several possible solutions to the preemption cycle problem. Paulussen et al. [50] propose a market mechanism to preserve individual rationality in the face of preemption (agents can pay one another to offset the costs of a preemption). However, if preempting agents consider only the total change in utility experienced by the preempted agent when determining how much to pay for a preemption, they create an *arbitrage* opportunity - a situation where shrewed agents can accumulate large amounts of money by selling resources for much more than the cost of replacing them. For example, suppose that Bob needs a

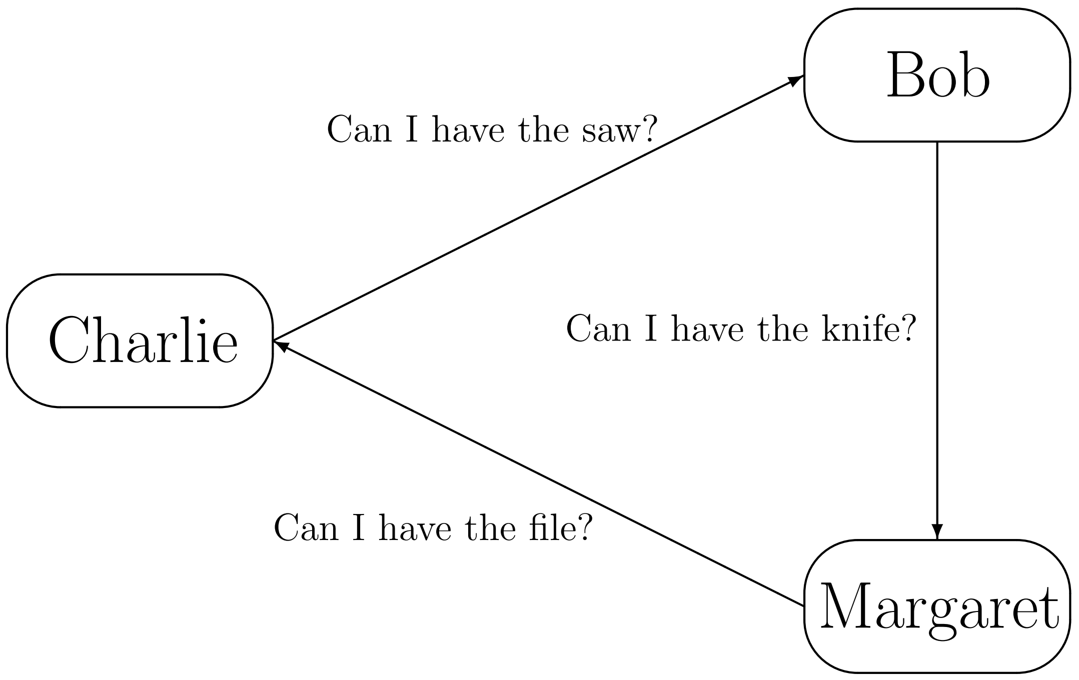


Figure 4.1: A visual depiction of the preemption cycle example.

saw or a file, and gets the same value for either. Margaret can only use the saw. Bob has the saw now, while the file is free. If Margaret pays Bob for the utility lost when he yields the saw and ends up with nothing, Bob can take the money, acquire the file for free, coming out well ahead. Another way of viewing this is that Bob is capturing a disproportionately large amount of the economic surplus gained by the preemption.

Paulussen et al. address this issue by considering the single best action a preempted agent could take *immediately*. Continuing with the example above, since the best action Bob can take is to acquire the file (costing him nothing), Margaret should pay Bob nothing at all when preempting the file. Though initially promising, this approach encounters problems when the best action the preempted agent can take is to initiate a preemption of its own, which can cascade into a preemption cycle. Paulussen et al.'s solution is to make the worstcase assumption that the second preempted agent ends up with no resource at all, and that the preempting agent will have to compensate it accordingly, but once again this creates opportunities for arbitrage (though such opportunities are both less frequent and less profitable than before). Further, since this approach will cause preempting agents to occasionally overpay, needy but poor agents may occasionally be denied resources which they *could* afford, were the worst-case cost estimate replaced with a more accurate one. Neither of these features is a desirable property of resource allocation systems.

An alternate solution is to forgo distributed a allocation system altogether, and instead adopt a centralized system. While in general resource allocation problems are NP-Hard (via reduction from multiprocessor scheduling) [76], which makes exact solutions just as difficult to obtain in a centralized setting as in a distributed one, high quality heuristic solutions are possible. There are a few drawbacks to such systems. First, centralized systems are often considered to be less reliable than decentralized ones [71] [40] [12] [70]. In a fully distributed system, a single agent can fail without taking down the entirety of the system. Additionally, communication channels must handle much larger amounts of data transfer. While a distributed system requires agents to exchange only those pieces of information which are needed locally, a centralized system needs many distributed sensors to send it up to date information, and so must be able to process large amounts of incoming information at once. Further, if any piece of information changes, a centralized system must update (or potentially scrap) an incomplete optimization problem, while a distributed system can simply adjust any agents who are actually impacted by the change. In summary, although a centralized system could be appropriate for use when the problem is small or evolving slowly over time (thus circumventing the preemption cycle problem) in very large or dynamic problems such systems may have difficulty adapting, or may have to be content to search for nearly optimal solutions instead of optimal ones.

A third potential solution worth mentioning is the use of explicit prioritization to ensure important tasks are solved even if this creates bad outcomes for others. The advantage of this approach is that it skirts the idea of cost-benefit analysis entirely by allowing for an explicit ranking of tasks. This approach is used in some heuristic bandwidth allocation systems for cellular networks [39], because of the existence of several broad categories that completely determine the priority of a task (e.g. real time voice conversations v.s. data transfers), and there being only one type of resource (bandwidth). This limited scope allows for the creation of a lexicographic ordering over tasks, which can be used to preempt the task with minimum rank. Notice however, that this also requires either a centralized system, or one in which all agents transmit information to all others quite frequently, since when an agent wants to perform a preemption it needs have complete information to determine the lexicographic ordering of tasks before doing so. Further, lower priority agents must relinquish their resources even when doing so brings only a small benefit to the preempting agent, at great cost to the preemptor, which may not be suitable in all problem domains.

4.3 Contingency Plans

The method of resolving preemption cycles proposed in this work is to have agents create 'contingency plans' when they acquire a resource. A contingency plan is a sequence of actions that the agent would take if the resource they hold now were taken from them. For example, if Bob acquires a saw, then Bob would immediately create a new contingency plan.

If contingency plans take the form of a TOC strategy, then it is possible to assign an expected value to the execution of the plan. Indeed, such a value is simply a byproduct of the optimization process: the expected gain in utility which would result from executing the plan. To obtain an estimate of the costs of preemption, an agent now needs only to compare his expected value for continuing to hold the resource to the expected value of his plan. For example, suppose that, after acquiring the saw, Bob builds a TOC strategy with the set $\{R - saw\}$ (that is, with all resources except the saw). The new strategy has an expected value of 16. When someone asks to preempt Bob's resource, Bob can now compare the benefit he would get by continuing to hold the resource against the benefit expected through the execution of the contingency plan. The difference between these two values is an estimate of the costs of preemption.

Naturally, there are both advantages and disadvantages to this approach. A drawback is that agents' plans will not be completely correct. For example, if at timestep 1, Bob makes a contingency plan which assumes that the file is not in use, then when Margaret takes the file during timestep 2, Bob's plan will now be too optimistic. We address this issue by having Bob use a simple learning algorithm to estimate the likelihood of his plan being overly optimistic. This is detailed in section 5.2.2. We are less concerned about the impact of agents producing plans which are too conservative, since longer plans, even when they are incorrect, will still tend to be less conservative than the single-step lookahead plans of prior authors [51].

Another potential drawback is that we have to do a large amount of additional computation. While agents in other systems [51] computed preemption costs only when a preemption attempt actually took place, agents in our system compute them every time they take a resource. Further, the more detailed preemption cost estimates used in our system can take considerably longer to produce than the simple ones of prior authors. We have two reasons for thinking these features are minor, rather than substantial, drawbacks. First, in the computation of a contingency plan, our agents receive at no extra computational cost, a plan of action which might allow them to improve their situation immediately. That is, if the contingency plan has an expected value in excess of the value of holding the agent's resource (preemption costs are negative), then the agent can start executing the plan immediately to obtain a better resource. Second, our system, though able to solve resource allocation problems in general, is intended to solve MARA problems which contain very high congestion - that is, problems where there are many more agents than there are resources. In these environments, the time spent computing a preemption cost estimate can actually be amortised over a large number of subsequent preemption events.

A final worry is that generating long contingency plans may not be a good use of computational resources, even if we can use learning algorithms to discount the value of overly optimistic plans. To address this concern, we incorporate a second learning process, also discussed in section 5.2.2, which estimates the 'dynamism' or 'churn' of the problem, that is, the learning process attempts to estimate the rate at which the present allocation of resources is changing. When this rate is high, agents dynamically reduce the length of their plans to compensate. When the rate is low, agents build longer plans instead. An alternate view of this is that agents are modelling the time horizon beyond which they are no longer able to predict the future effectively using local information alone.

Having addressed these drawbacks we now consider the advantages of using contingency plans. First, as mentioned above, once one has a contingency plan, one can act on it if doing so might result in getting a better resource. Further, one can compute estimates of preemption cost in constant time once the plan has been generated, allowing agents to answer many preemption requests very quickly. This attribute is especially useful in a fully distributed system. If agents are physically deployed on hundreds of small, computationally underpowered devices (smart phones for example), then an agent which must perform excessive computation or send and receive several messages to process a preemption request will probably not be able to function well during periods of high congestion when it might have to process dozens or hundreds of such requests per second.

An additional advantage is that the system can be fully distributed. Assuming an agent knows its own utility function, agents only need to communicate with other agents during planning (when the agent needs to verify the preemption costs associated with acquiring various other resources), and during preemption requests. This produces an amortized transmission cost of $O(R/k)$ transmissions per agent per timestep, which is not an unreasonable rate, especially in problems with large amounts of congestion and little churn, where $O(R/k)$ will be small.

Chapter 5

Multiagent Resource Allocation With Direct Preemption

This chapter contains the main application of the research accomplished in this work: the realization of a fully distributed resource allocation system with reasoned preemptions. The system has three main components, each of which is described in some detail to begin with. After describing the three components and their interactions, we present the results of several benchmarking experiments performed using a simulation of an abstracted resource allocation problem. We draw the reader's attention additionally to the two techniques which allow our solution to overcome the preemption cycle problem discussed in chapter 4. First, we utilize TOC strategies as the basis of agents' plans in the system, and use the valuations of these strategies to estimate the opportunity costs of a preemption. Second, we use a pair of simple learning algorithms to estimate and account for the errors in estimation produced by using this technique. Both of these properties of the system are discussed in section 5.2.2. Additionally, resource proxy agents play a major role in the system, providing coordination and user modelling components, and are discussed in a subsection of section 5.2.3.

5.1 System Components

The first component used by our system is a model of resources. In this model, a resource r is given by three properties:

1. A type θ_r, which will determine what sort of jobs the resource can process, and/or how efficiently it can process them.

2. A Bother-so-Far value BSF, indicating how bothered the resource is.

3. A current (possibly non-existent) task that the agent is processing, *r.current*.

Resources are each assigned an agent inspired by Cheng, Fleming and Cohen's type IV proxy agents for bother cost minimization [5] [10]. We refer to this as a resource's proxy agent throughout. As will be explained in detail in section 5.2.3, proxy agents serve to enable effective coordination of planned TOC strategies among all the agents in the system. This is achieved through the responses that proxy agents provide when asked by agents whether their planned transfers of control to particular resources are likely to succeed, based on the proxies' knowledge of the resources' current states.

Resources exist to process tasks. Our model of a task t consists of:

1. A type θ_t, which determines which resources the task will find useful.

2. A current (possibly non-existent) resource which is presently processing the task *t.current*. Note that, within the system it is always the case that if $t.current = r$, $r.current = t$ for a task t and resource r.

3. A measure of the time this task has spent in the system *t.timewaited*, which can be used to model increasing costs associated with longer wait times.

4. A measure of remaining processing time *t.processing_time_remaining*, which can be ∞ if *t.current* does not exist.

Tasks have much more powerful agents than resources do. For convenience, we often refer to these simply as "Agents" within the context of the system, rather than "task agents". Resource agents will always be referred to as "proxy agents" or "resource proxy agents".

An agent a in our system has:

1. A task *a.task* which the agent is attempting to have processed by a resource.

2. A plan, *a.plan*, which is a sequence of resources representing the agent's intentions for some number of future time steps.

3. A measure of the expected utility of its plan $a.eu_{plan}$.

4. A measure of the time since the agent last updated its plan $a.t_{update}$.

5. An internal learning model which attempts to dynamically optimize the plan length the agent to uses, *a.plan_length_model*, discussed in detail later in this chapter.

6. An internal learning model which attempts to dynamically optimize the agents' "expectation of preemption", *a.preemption_model* a property discussed at length later in this chapter.

In addition to the above building blocks, the system requires the specification of the following **problem specific functions**:

1. $processing\_time(\theta_r, \theta_t)$, which produces the total number of time steps required for a resource with type θ_r to process a task of type θ_t, assuming that no interruptions occur. This function must also produce special values when passed $processing\_time('useless', \theta_t)$, which indicate the expected processing time for a task with no resource. This can be infinite if and only if EU below can process an infinite value for the *old_time* parameter.

2. $EU(new\_time, old\_time, \theta_t, time\_waited)$, which produces the expected utility of completing the processing of a task with type θ_t at *new_time* instead of *old_time*, given that the task has been in the system for *time_waited* time steps already.

3. $P_{stop}(EU\_net, BSF)$, which produces the probability that a resource with the supplied bother-so-far value would agree to process a job which produced a net improvement in expected utility of EU_net over continuing with its current job instead.

4. $bother\_increase\_function(EU\_net, BSF, \theta_r)$, which supplies the increase in bother expected when a resource with type θ_r and present bother of BSF is bothered with a request that would produce a net improvement in utility of EU_net.

As a result of the above specification, a problem instance which the system can process needs the following user-supplied definitions, which we will assume the existence of throughout the chapter:

1. A set R of resources, each of which is given a type, and an initial bother value (often zero).

2. A set T of tasks, each of which is given a type, and which may be modelled as having waited for some time already.
3. A specified number of agents which can be used in the system (possibly less than the number of tasks).
4. A function $priority\_job(T)$, which extracts the most important job from T for an unoccupied agent to work on[1]
5. Definitions for all four problem specific functions ($processing\_time$,EU, P_{stop}, and $bother\_increase\_function$).

This definition is quite similar to the conventional one used in distributed resource allocation work [82], but incorporates several user modelling components (for example, bother, and a P_{stop}), which are not found in the standard definition. We believe that the inclusion of such components is necessary to model a dynamic, real world allocation system, in contrast to the static system operating on computational agents and inanimate resources usually assumed by multiagent resource allocation work [51] [82] [8].

5.1.1 Example

Suppose we wished to model a resource allocation problem for the tasks of a plumbing company, *AgentPipes*. The company has 10 employees. One is a master plumber, 3 are journeymen, and 6 are apprentices. Today, the plumbers have 20 jobs to do between them: Seven dishwasher installations, 11 leaky sinks, and two flooded basements. In addition, 19 of the jobs were just called in this morning, while one of the flooded basements is spillover work from the day before. Suppose that getting to a job from anywhere in town takes an expected travel time of one hour. Further, suppose that table 5.1 gives the time expected for a plumber of each type to repair each type of job.

Suppose that AgentPipes has a highly exceptional devotion to customer satisfaction, and tries to minimize customer discomfort rather than maximizing profits. The company believes that every hour spent with a leaky sink is worth, on average, $-\$20$ to a customer. Similarly, an hour with an yet to be installed dishwasher as an expected value of $-\$40$, while every hour with a flooded basement is worth $-\$500$. Finally, we get a rather vague

[1]Note: A purely random prioritization function appears to outperform most naive deterministic orderings, suggesting that writing a 'good' prioritization function is a hard resource allocation problem in and of itself.

$\theta_r/\theta t$	Dishwasher Installation	Leaky Sink	Flooded Basement
Apprentice	5 hours	1 hour	24 hours
Journeyman	2 hours	1 hour	12 hours
Master	1 hour	1 hour	4 hours

Table 5.1: A description of the time taken by different grades of plumber to complete various plumbing tasks in the AgentPipes example

description of bother costs, which states that employees will resent being distracted from important work, and that more experienced employees will resent it exponentially more than apprentices.

In this example, we define the problem formally as follows:

- R is a set of 10 resources. One has type *master*, 3 have type *journeyman*, and 6 have type *apprentice*. All resources have an initial bother value of $BSF = 0$.
- T is a set of 20 tasks. 7 have type *dishwasher*, 11 have type *sink* and two have type *basement*. All jobs are assumed to have initial wait times of zero, except one of the basements, which has an initial wait time of 24 hours.
- As no limitations were placed on computational resources, we assume at most 20 agents (one for each job) will exist in the system at once.
- Since all tasks will be assigned an agent, and all agents are identical, the *priority_job* function can be set arbitrarily.
- The processing time of a job is just the value from table 5.1, plus the expected one hour of travel associated with each task. So, $processing\_time(\theta_r, \theta_t)$ is just the corresponding value from that table plus one hour.
- The expected utility of reducing the expected time for the completion of a job is given by:

$$EU(new\_time, old\_time, \theta_t, time\_waited) =$$

$$\begin{cases} 10(old\_time - new\_time) & if\ \theta_t = sink \\ 20(old\_time - new\_time) & if\ \theta_t = dishwasher \\ 500(old\_time - new\_time) & if\ \theta_t = basement \end{cases}$$

- The probability that a plumber responds to a request to switch jobs (by picking up his phone) is given by perhaps:

$$P_{stop}(EU\_net, BSF) = \begin{cases} 0.99 & BSF < 50 \\ 0.05 & BSF \geq 50 \end{cases}$$

- And the bother increase function might be given by:

$$bother\_increase\_function(EU\_net, BSF, \theta r) =$$

$$max(0, e^{-EU\_net} \times \begin{cases} 0.5 & if\ \theta_r = apprentice \\ 1 & if\ \theta_r = journeyman) \\ 4 & if\ \theta_r = master \end{cases}$$

5.2 The Algorithm

This section describes the proposed new algorithm for solving resource allocation problems. The algorithm is first specified at a high level, in terms of subroutines associated with each of the primary system components. These subroutines are then expanded upon in detail. We conclude the section with an example showing how the system can solve a simple resource allocation problem.

The core of our new framework lies in its negotiation protocol, presented below as part of the behavior of an agent throughout:

1. Let A be an agent representing a particular task $A.t$.
2. Let P be a plan of action for A which specifies the optimal (or near optimal) order for A to request resources.
3. Let p be the resource suggested for acquisition at the current time by P, and t_{old} be the completion time for $A.t$ without the specified resource.
4. If p is unowned, A acquires it.
5. Otherwise, if p is owned by agent B, compare the gain made by A preempting p with the TOC-based **preemption cost estimate** produced by B.

6. If the gain exceeds the cost, A takes p, and B now searches for a new resource based on its own plan of action.
7. Otherwise. If p is the last step in P, generate a new plan.
8. Otherwise, remove p from P, and GOTO step 3.

This process is specified more formally below, but referring to the specification above may assist the reader in understanding the behavior of the formal specification. For now however, we draw the reader's attention to item 5. Recall that we previously mentioned the importance of both TOC strategies and agent learning algorithms to our solutions. It is the **preemption cost estimate** mentioned above which forms the primary point at which these two features come together to provide a solution. TOC strategies allow an estimate to be made, while learning algorithms help agents to make those estimates more accurate. As will be explained in detail in section 5.2.2, agents seek to learn both about the optimal plan length and about the demand for similar resources from other agents (i.e. the congestion).

5.2.1 The Main Loop

The algorithm's main loop is described in algorithm 5.1. This is the portion of the algorithm which is called to initiate solving of a new resource allocation problem. Essentially, given the problem specification from section 5.1, this algorithm initializes the set of agents, assigns them to tasks according to the problem specification, and then iteratively updates the agents, resources, and tasks as the system steps forward in time, until all tasks have been processed. ***Nota bene:*** certain minor specifics of the algorithm have been omitted or obfuscated to make it fit on a single page. In particular, we do not specify what to do when *priority_job* returns no task (due to the passed subset of T being empty). Consequently, on line 20, agents should only step if they have a task. Additionally, when a new agent is created on line 9, the agent should be assumed to have knowledge of both k, the maximum permitted length of its plan, and contact information for R, the set of all resources in the simulation.

5.2.2 Agent Behaviors

The behavior of agents at each timestep is specified by the *step_agent* function, given in algorithm 5.2. The behavior of agents at each step consists of four operations. First, the

Algorithm 5.1 The main loop, called to begin processing a new resource allocation task in the proposed distributed resource allocation system.

```
1: LET $n$ be the maximum number of agents allowed in the system at once.
2: LET $T$ be the set of tasks to be addressed by the system.
3: LET $R$ be the set of resources.
4: LET $priority\_job : \mathbb{P}(T) \rightarrow T$ be as described in section 5.1
5: LET $processing\_time$, $EU$, $P_{stop}$, and $bother\_increase\_function$ be as in sec. 3.2
6: LET $k$ be the maximum plan length agents are permitted to generate.
7: LET $A = \emptyset$, $T_{complete} = \emptyset$ and $T_{added} = \emptyset$
   {Generate the initial agents in the system, and assign them to the most urgent tasks}
8: while $|A| < n$ do
9:   LET $a$ be an agent, $a.task \leftarrow priority\_job(T/T_{added})$
10:  SET $A \leftarrow A \cup a$
11:  SET $T_{added} \leftarrow T_{added} \cup priority\_job(T/T_{added})$
12: end while
   {Until all tasks have been solved...}
13: while $|T - T\_complete| > 0$ do
     {Each agent takes a step, see algorithm 5.2}
14:  LET $A'$ be a random permutation over $A$.
15:  for all $a \in A'$ do
16:    if $a.task \in T_{complete}$ then
         {If the agent's task is complete, assign it a new task.}
17:      SET $a.task \leftarrow priority\_job(T - T_{added})$
18:      SET $T_{added} \leftarrow T_{added} \cup priority\_job(T - T_{added})$
19:    end if
20:    $step\_agent(a)$
21:  end for
     {Then, each resource takes a step, see algorithm 5.5}
22:  for all $r \in R$ do
23:    $step\_resource(r)$
24:  end for
     {Finally, each task takes a step, see section 5.2.4}
25:  for all $t \in T$ do
26:    $step\_task(t)$
27:    if $processing\_complete?(t)$ then
28:      SET $T_{complete} \leftarrow T_{complete} \cup t$
29:    end if
30:  end for
31: end while
```

agents update their internal congestion models, learning about the frequency of congestion in the world around them. In this context, congestion is used to estimate the relative demand for resources like those a particular agent might be interested in acquiring. For example, if an agent wants a machine, its behavior should differ between the cases where there is 1 machine for every agent who wants one, and one machine for every 20 agents who want one. Second, the agents generate a new plan if their current plan is empty. Third, the agent looks at the next resource in its plan. It asks that resource's proxy agent whether that requesting service is a good idea (i.e. one that would produce a net increase in the utility of the overall allocation) or not. If it is not, the agent generates a new plan, and repeats the contact with the proxy agent for the first resource in this new plan (though it does not generate an additional new plan if this second resource is also not worth asking). If the proxy agent confirms that it is a good idea to contact the resource now, the agent asks the resource to drop anything it might be doing and start processing the agent's task. If the resource agrees, the agent then generates a new contingency plan. Finally, the agent discards the front element of its plan, and updates its estimated value for executing the entire plan accordingly.

Notice that some portions of the algorithm have the potential for more efficient implementations than are realized in our system. For instance, the process of updating the expected value of a plan at the end of each time step is written as a full valuation using the recurrence relation from equation 3.2, which takes time equal to the plan length. If we stored the contributions of each element of the plan at the time of generation, we could use only a linear number of steps in the plan length to valuate it over the lifetime of the plan, instead of the quadratic (linear number of linear time operations) amount of time required as written. This inefficient method was selected both to simplify presentation and to hasten initial implementations. However, the implementation used later in this chapter for simulations utilizes many small optimizations of this sort in practice.

The plan generation process detailed in algorithm 5.3 is essentially a wrapper for whichever TOC strategy generation method the user prefers (see chapter 3 for a discussion about why different generation algorithms might be preferred). First, the length of the plan is determined by the agent's internal congestion model. This is required because both over-planning (generating plans that are too detailed, and which prevent agents from adapting to rapidly changing circumstances) and under-planning (generating plans which are too short or too vague, and thus fail to produce optimal decisions), have been shown to impact the performance of TOC-based systems [63]. Indeed, this is confirmed by our own experiments later in this chapter.

In addition, once we have generated a new plan, the agent's plan length model is flagged so that it can be updated to reflect a strategy regeneration taking place. The new TOC

Algorithm 5.2 An algorithm describing the behaviors of agents at each time step in the main loop.

```
1: $step\_agent(a)$
2: Assume that $a$ has contact information for the proxy agents of a set of resources $R$
3: Assume that $a$ has knowledge of a maximum plan length $k$.
4: Update $a.preemption\_model$ (i.e. congestion model) & $a.plan\_length\_model$ (alg. 5.4)


   {If the agent doesn't have a plan, make one.}
5: if $a.plan$ does not exist, or $|a.plan| = 0$ then
6:     $a.plan \leftarrow generate\_plan(a, R/a.task.current)$ (see algorithm 5.3)
7:     $a.eu\_plan \leftarrow tocValuation(a.plan)$ (see equation 3.2)
8: end if
   {Check whether the next step in the plan would presently produce a net increase in
   expected utility.}
9: LET $r$ be the next resource in $a.plan$ (n.b. could be 'wait' from section 3.2.3)
10: if $allow\_transfer\_attempt?(r.proxy, a.task)$ (see algorithm 5.7) then
        {If executing the next step in the plan is a bad idea, make a new plan.}
11:     $a.plan \leftarrow generate\_plan(a, R/a.task.current)$ (see algorithm 5.3)
12:     $a.eu\_plan \leftarrow tocValuation(a.plan)$ (see equation 3.2)
13:     LET $r$ be the next resource in $a.plan$ (n.b. could be 'wait' from section 3.2.3)
14: end if
    {If executing the next step in the (possibly revised) plan will produce a net increase
    in expected utility.}
15: if $allow\_transfer\_cost(r.proxy, a.task)$ (see algorithm 5.7) then
        {Ask the resource to process our task.}
16:     $request\_transfer(r, a.task)$ {If it agrees, then make a contingency plan.}
17:     if $a.task.current = r$ then
18:         $a.plan \leftarrow generate\_plan(a, R/a.task.current)$ (see algorithm 5.3)
19:         $a.eu\_plan \leftarrow tocValuation(a.plan)$ (see equation 3.2)
20:         $a.preemption\_model.expected = TRUE$
21:     end if
22: end if
    {Remove the front element from $a.plan$}
23: $a.plan \leftarrow rest(a.plan)$
24: $a.eu\_plan \leftarrow tocValuation(a.plan)$ (see equation 3.2)
```

strategy is then returned.

Algorithm 5.3 An algorithm describing task agents' plan generation methodology

```
1: $generate\_plan(a, R)$:
2: LET $plan\_length \leftarrow a.plan\_length\_model.predict()$ (See equation 5.3)
3: LET $method(O, k, V, P_{continue}, t, \Delta t)$ be one of the four plan generation methods from
   chapter 3.
4: SET $a.plan\_length\_model.updated\_plan \leftarrow TRUE$
5: return $method(R, plan\_length, compute\_preemption\_costs, (1 - P_{stop}), 0, 1)$
```

Nota Bene: The expected utility function which is passed to the plan generation method is **not** the EU function provided as part of the problem definition. Instead, we provide *compute_preemption_costs*. This function is used by resource proxy agents to compute a $EU_{preempt}$ estimate, from equation 4.1 in chapter 4. This equation is a function of EU, and resource proxy agents (see section 5.2.3) have all the information required to compute it when provided with the preempting task. Thus, an agent using *generate_plan* that wishes to determine the expected value of asking for a resource r at a particular time can ask the proxy agent of r to compute it with respect to the system agent's task by calling $compute\_preemption\_costs(r, a.task)$. The proxy agent uses its local information in conjunction with the type of *a.task* to estimate the total net change in the expected utility of the overall allocation from making the specified TOC request, and sends it back to the agent who can then continue to generate its plan.[2]

Agent Learning

As touched upon briefly in the proceeding subsection, agents in this system will tend to experience problems when their plans are too long or too short relative to the rate at which circumstances are changing within the allocation task. It is insufficient to have a global model of the rates of change because individual agent's experiences may vary widely. For example, if 500 agents are suddenly assigned tasks which can be solved using 200 resources, they will tend to need short plans, since the circumstances of the allocation problem will be undergoing rapid changes. In contrast, if a small number of agents are competing for a particular type of resource, and have many resources to choose from, but each has only a low probability of agreeing to process an agent's task, then the agents will tend to derive greater utility from making longer term plans. Since, in a resource allocation problem with

[2]Note that we could cache values to limit redundant communication if necessary.

several different types of resources and tasks, both of the circumstances can be present at once, individualized models are required.

The model we use is based on a simple exponentially decaying memory, similar to a capacitor or to the bother cost modelling used by Cheng et al. [10]. In this model, we track the degree of churn in the resource allocation problem by adding 'charge' (i.e. a new interaction cost) to the memory each time our plan is regenerated, which usually takes place when our expectations about the future were incorrect in a damaging way. Then, at each time step, the charge 'leaks' from the memory, so that the agent slowly forgets about past churn events. The upshot is that, when there is lots of churn in the problem, agents will quickly adapt and make shorter plans. When churn dies down, they will gradually forget about past churn events until their plans become long enough to encounter churn again. This model is described by equation 5.1, where I is the set of all instances where the agent's plan was regenerated (all occasions where $generate\_plan(a, R)$ was called), t_i is the length of time in the past at which i, a regeneration, took place, and α and c_i are problem specific parameters which describe the rate at which agents forget previous regeneration events and the cost of event i respectively. Notice that α must lie in the range of 0 (instantly forgetting interactions) and 1 (never forgetting interactions). A higher relative value c_i allows the model to adapt faster to increases in problem churn, and will tend to produce shorter plans in general. Higher values of α allow the model to forget previous churn faster, causing plan lengths to be longer in general, and to rapidly return to their maximum lengths when churn is absent from the problem.

$$plan\_length\_model.value = \sum_{i \in I} \alpha^{t_i} \times c_i \tag{5.1}$$

For example, imagine an agent Bob who is in a hospital. Bob needs to get a machine (resource) for his patient's treatment (task). Initially, Bob might suppose that no one else is in the hospital. Bob builds a plan of length 5 based around this assumption and then goes about executing it; however, after several minutes, Bob's information becomes incorrect and in fact the hospital is now quite busy. When Bob contacts the proxy agent for the next machine, he is informed that actually the machine is in use by a much more urgent patient, and that transferring the machine to Bob's patient is not a good idea after all. This event become a negative interaction in Bob's plan length model, because the environment changed dramatically before he updated his plan. Bob adds the interaction cost (which we assume is just 1) to his model, and then makes a new plan of shorter length. If several timesteps pass before Bob has another negative interaction of the sort described above, then Bob's next plan may be of length 5 again. If, on the other hand, Bob has many more interactions he may accumulate enough interactions to stop planning ahead at all, and will just make locally optimal decisions at each timestep.

In a similar fashion, our agents learn about the likelihood of a preemption taking place using the model in equation 5.2. This equation also uses a capacitive memory, but here a new value of d_i is added each time the agent's resource is preempted. This is measured by checking whether the agent's task had a resource in the previous time step, but does not have one now. γ takes on the same role as α in the plan length model, with higher values of γ (which again, ranges between 0 and 1) causing the agent to be scarred by preemption events for a long time, while smaller values make the agent optimistic that the frequency of future preemption events will be smaller than the frequency of such events at present. d takes on the role of c from the plan length model, representing the cost of an individual preemption. In the system as presented in this document, we adopt a very simple model for estimating the value of d (we assumed that $d = 1$ at all times). However, it should be possible to produce a more nuanced model by using the ability of proxy agents to estimate the improvement in net utility resulting from a preemption (see section 5.2.3). In this model, preemptions which produce relatively larger improvements in the overall system utility will tend to produce lower increases in the agent's preemption model, since these preemptions are better justified and therefore easier for the agent to accept than preemptions which produce only a marginal gain in utility.

$$preemption\_model.value = \sum_{i \in I} \gamma^{t_i} \times d_i \tag{5.2}$$

Following the example above, let's suppose that Bob takes a machine while still under the mistaken belief that there is no one else in the hospital. Bob builds a contingency plan which describes what he would do if someone else took his machine. Since the plan is built using Bob's mistaken belief, it has a high value (if no one else is in the hospital, it will be easy to get a different machine if someone needs Bob's). When another agent tries to preempt Bob's machine, Bob counts this as a preemption model interaction. Regardless of the outcome of the preemption, Bob will discount the value of contingency plans made in the immediate future, since preemption attempts are indications that there are no free resources for people like Bob's patients, making it harder to acquire them. Further preemption attempts will cause Bob to produce even deeper discounts in his valuations, while long stretches without any such attempts will bring his valuations back up to full value.

The values stored in the agent's models are updated by line four in the *agent_step* function (algorithm 5.2). We describe this update step in algorithm 5.4. Essentially, the algorithm assumes that *a.plan_length_model.value* and *a.preemption_model.value* store the model values from the previous time step, and that α and γ are known. At each step, we first update the exponents of α and γ respectively in each element of I, and then add any new interaction costs to the models' values. We also reset the model flags which are used by algorithms 5.3 and 5.2 to signal the need for an update.

Algorithm 5.4 An algorithm describing the learning process for the agent's internal congestion models.

```
1: $update\_models(a)$:
2: SET $a.plan\_length\_model.value \leftarrow a.plan\_length\_model.value \times \alpha$
   {If our plan was updated in the last time step (see algorithm 5.3, line 4)}
3: if $a.plan\_length\_model.updated\_plan = TRUE$ then
4:    LET $c$ be the cost of a plan regeneration interaction in the system at present.
5:    SET $a.plan\_length\_model.value \leftarrow a.plan\_length\_model.value + c$
6:    $a.plan\_length\_model.updated\_plan = FALSE$
7: end if
8: SET $a.preemption\_model.value \leftarrow a.preemption\_model.value \times \gamma$
   {If our resource was preempted in the previous time step (see algorithm 5.2, line 20)}
9: if $a.task.current = \emptyset$ AND $a.preemption\_model.expected = TRUE$ then
10:    LET $d$ be the cost of a preemption interaction in the system at present.
11:    SET $a.preemption\_model.value \leftarrow a.preemption\_model.value + d$
12:    $a.preemption\_model.expected = FALSE$
13: end if
```

A careful reader will notice that one final component of the agent learning models remains unspecified, namely the *predict* function used on line 2 of algorithm 5.3. The prediction function uses the values stored by the models to produce an estimate of the ideal length of plan to generate (for the plan length model) or of the coefficient the agent should use to adjust its estimate of actually acquiring and holding a new resource when reporting its estimated preemption costs (which is used by resource proxy agents to valuate the costs of a preemption, see section 5.2.3). The plan length model prediction is given by equation 5.3. When the model has a value of zero, this function returns k, the maximum allowable plan length. As the model's internal value grows, the returned value goes to 1, the minimum possible plan length, where the agent is generating a new plan at every step, and just making greedy, locally optimal decisions. A plot of the returned plan length model prediction for $k = 15$ is shown in figure 5.1.

$$a.plan\_length\_model.predict = \lfloor \frac{k-1}{1 + a.plan\_length\_model.value} \rfloor + 1 \tag{5.3}$$

The model used to predict the preemption cost coefficient is similar, but not identical. It is shown in equation 5.4, with an example plot of the resulting function in figure 5.2. Notice especially that that preemption model's predicted value is not discrete. Again, the function is a simple rational equation. When an agent's resource has never been preempted, then the function returns a value of 1, meaning the agent does not discount the value of its contingency plan at all when responding to a request for preemption. As the preemption model's value increase, the agent rapidly becomes conservative, and starts to assume that, even if it can get a new resource following a preemption, that resource will be preempted as surely as its current one was. Consequently, the agent begins discounting the expected value of its contingency plan, with the coefficient to the expected value of the plan going to zero as the agent's preemption model value increases.

$$a.preemption\_model.predict = \frac{1}{1 + a.preemption\_model.value} \tag{5.4}$$

Examples

A brief example may help cement the ideas of this subsection. Suppose that an agent *Bob* forgets 5% of the effects of previous interactions every hour (so $\gamma = \alpha = 0.95$). Additionally, Bob learns at a fixed rate of 1 unit per interaction ($c = d = 1$), and has a maximum plan length of $k = 5$. Three hours ago, while Bob was trying to work with the saw, Margaret took it from him. Bob's contingency plan called for him to take the knife next, but it turned out that his information was out of date, and Charlie (who has a more important task than Bob's) was already using the knife. Bob spent an hour making a new plan (he thinks slowly...) and then decided to ask Joe for the axe. Unfortunately, while Bob was thinking, Charlie finished with the knife, and has taken the axe. Bob's information is once again out of date! He quickly generates a new, shorter plan to take the now unused knife, which succeeds. Bob uses the knife for an hour, but then Charlie needs it again and takes it from him[3].

The progression of events is summarized as follows:

- Three hours ago, Bob's resource was preempted, and then his plan turned out to be out of date.
- Two hours ago, Bob's new plan was out of date again.

[3]We assume that Bob has to start the task over if his resource is preempted mid-task.

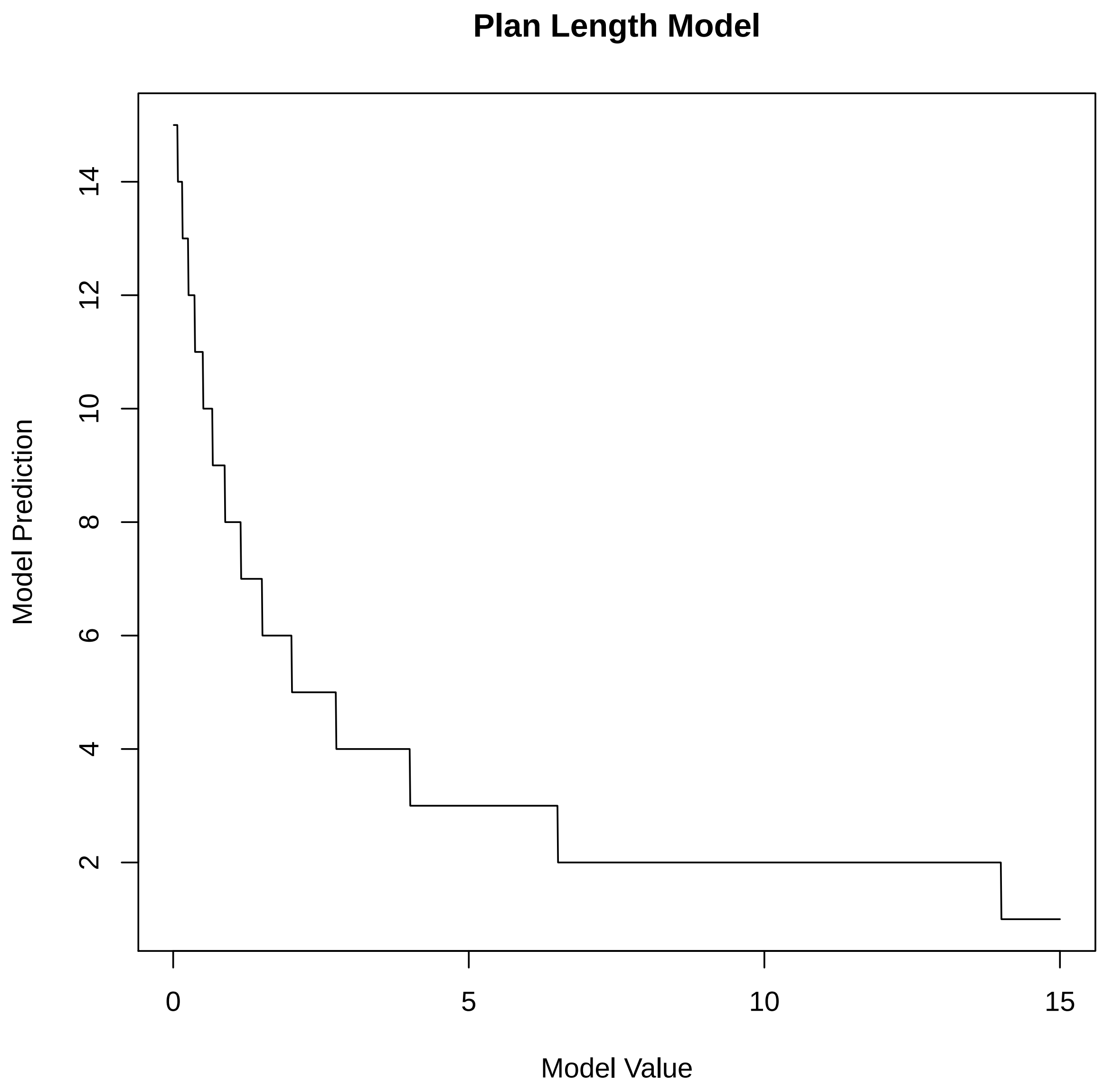


Figure 5.1: An example of predicted optimal plan length under the model discussed in equation 5.3, with $k = 15$.

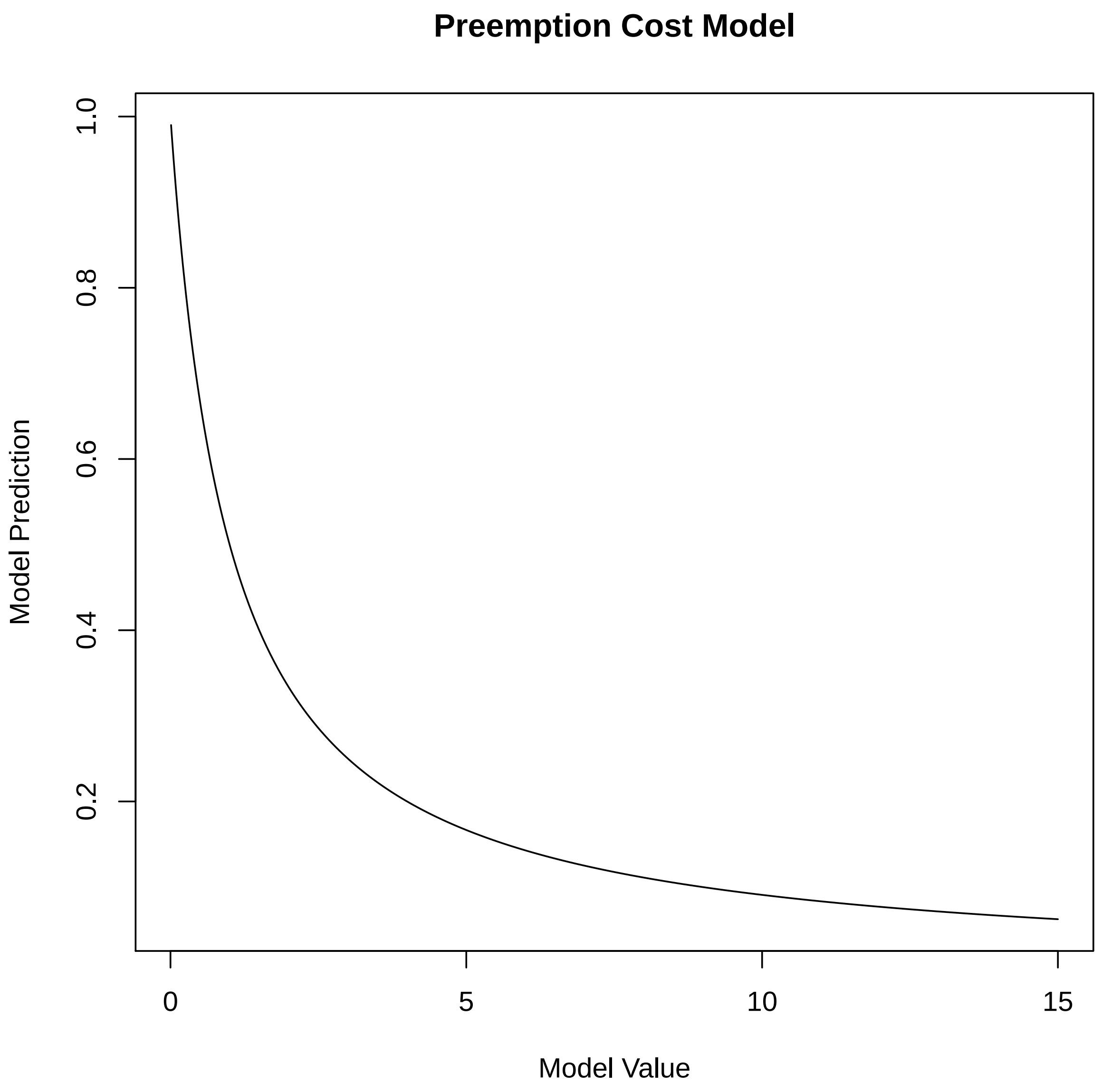


Figure 5.2: An example of predicted preemption cost coefficient under the model discussed in equation 5.4

Time	$plm.value$	$plm.predict()$	$preempt.value$	$preempt.predict()$
t-3	1	$\lfloor\frac{5}{2}\rfloor + 1 = 3$	1	0.5
t-2	$1 \times 0.95 + 1 = 1.95$	$\lfloor\frac{5}{2.95}\rfloor + 1 = 2$	$1 \times 0.95 = 0.95$	$\frac{1}{1.95} = 0.513$
t-1	$1.95 \times 0.95 = 1.85$	$\lfloor\frac{5}{2.85}\rfloor + 1 = 2$	$0.95 \times 0.95 + 1 = 1.9$	$\frac{1}{2.9} = 0.345$
t	$1.85 \times 0.95 = 1.76$	$\lfloor\frac{5}{2.76}\rfloor + 1 = 2$	$1.9 \times 0.95 = 1.81$	$\frac{1}{2.81} = 0.356$

- One hour ago, Bob's resource was preempted again.

Assuming that Bob had initial model values of zero, then then the values and predictions of his models as time passes are summarized in table 5.2.2.

Discussion

It is important to note that we make no claims regarding the optimality of the presented learning models for the roles assigned them. In fact, more detailed models offer a very thought provoking avenue of future research. For example, a more formal statistical model using Bayesian methods [3] might yield a substantial improvement in algorithm performance. It might also be interesting to incorporate active learning into the model [69], so that agents can deliberately issue queries about the parts of their model which may be unreliable. Finally, for non-idealized applications, the use of model-free regression [64] might produce a more robust system than any specific learning model can provide.

5.2.3 Resource Behaviors

Here we present the algorithms governing behaviors of resources in our system. These behaviors are produced by the calls at line 23 of the 'main loop' in algorithm 5.1, but also whenever an agent's request for processing of a task makes it past the resource's proxy agent, as at lines 15 and 16 of the agent stepping behavior (algorithm 5.2).

First, we present the behavior of a resource in the course of normal operations, in algorithm 5.5. Then, we show the behavior of a resource when responding to a request for processing, in algorithm 5.6. Finally, we show and discuss the behaviors of resource proxy agents in algorithm 5.7. This last section is especially important, since proxy agents are responsible for estimating the expected utility of transferring a resource from its current task (if any) to a new task, a vital part of our proposed system.

Primary Resource Behaviors

To begin with, we describe the behavior of resources in updating their internal states at each timestep. A resource's behavior during a timestep update is relatively simple. If the resource is not processing a task right now, all that happens is an update of its internal bother model. Otherwise, we update the bother model and the time remaining until processing of the task is complete. Finally, if task processing is complete now, then the resource sets its current task to be undefined, so that agents will know they are once again free to accept new work. These steps are presented formally in algorithm 5.5.

Algorithm 5.5 An algorithm describing the behaviors of resources at each time step in the main loop.

```
$resource\_step(r)$ : {Update the model of the resource's bother, a learned measure of congestion with respect to this resource}
$update\_bother\_model(r)$ {If the resource is processing some task, decrease the processing time remaining.}
**if** $defined(r.current)$ **then**
    $r.current.processing\_time\_remaining \leftarrow r.current.processing\_time\_remaining - 1$
**end if**{If the resource has just finished processing the task...}
**if** $defined(r.current)$ **and** $r.current.processing\_time\_remaining = 0$ **then**
    {Then make the resource appear free again.}
    $r \rightarrow current \leftarrow undefined$
**end if**
```

An additional resource behavior is the protocol for actually initiating a preemption request. This behavior serves as the catalyst for a number of state changes between resources and tasks. In essence, the process first updates the resource's bother model and determines whether the resource will accept the preemption request. In simulation, this is done according to a random draw and P_{stop}, while in a deployed system, it would consist of actually asking the targeted resource whether or not it will switch tasks. If the resource agrees, then any task it is currently working on is informed of the preemption and discarded. Then the preempting task informs any resource it already possesses of the preemption, and discards the resource. Both the contacted resource and the preempting task now set their 'current' values to each other, and a new processing time is computed. The algorithm returns TRUE if if the preemption is a success, and false otherwise. This process is described formally in algorithm 5.6.

Algorithm 5.6 An algorithm describing the behaviors of resources in response to a request for preemption.

```
$request_transfer(r, task)$
LET $net\_EU = compute\_preemption\_cost(r, task, 0)$
{Update our bother model because the resource has actually seen a new request now.}
SET $r.bother\_model.value \leftarrow r.bother\_model.value + bother\_increase\_function(net\_EU, r.bother\_model.value, \theta_r)$
LET $draw$ be a random draw in $U(0, 1)$.
if $draw \leq P_{stop}(net\_EU, r.bother\_model.value)$ then
  {Alert the resource's old task to the preemption}
  if exists($r.current$) then
    SET $r.current.current = NULL$
    SET $r.current.processing\_time\_remaining = \infty$
  end if
  {Update who is processing what.}
  SET $task.current.current = NULL$
  SET $r.current = task$
  SET $task.current = r$
  {Update the processing times}
  SET $task.processing\_time\_remaining = processing\_time(r, task)$
  return TRUE
else
  return FALSE
end if
```

Bother Costs

In our system, resources are assigned proxy agents which store various pieces of information about them. One of these pieces of information is a model of the toll taken by frequent interactions with the system agents. The necessity of this model becomes apparent when one imagines a human being as a resource. While robotic resources might happily process thousands of requests from agents per hour, a human agent is more likely to disengage entirely from such a system, perhaps by shutting off their phone, or otherwise blocking out system alerts. The consequences of such a hiatus can be severe, and thus, a coordination system is required to prevent them.

Coordination is accomplished by having a proxy agent track interactions between the resource and system agents. The proxy agent assumes an exponentially rising cost for increasing the frequency of interactions, and comparatively faster or slower increases when interactions are more or less urgent. Our approach closely follows that of Cohen, Cheng and Fleming's work incorporating the costs of making humans interact with software into considerations about whether or not to contact them [10], but abstracts some of the modelling so as to support a more diverse set of bother functions.

In keeping with Cohen et al.'s approach, our bother model centers around a value referred to as 'Bother So Far' or BSF. BSF is an exponentially weighted moving average of the costs produced by previous interactions between a resource and the rest of the system, given by equation 5.5, where I is the set of all previous such interactions, $c(i)$ is the cost of interaction i, $t(i)$ is the time since interaction i took place, and β is a problem (or resource) specific parameter which represents the ability of a resource to remember the bother associated with previous interactions. In general, $0 \leq \beta \leq 1$, where $\beta = 1$ indicates that a resource remembers the costs of prior interactions forever, while $\beta = 0$ would entail forgetting them immediately.

$$r.bother\_model.value = BSF = \sum_{i \in I} c(i) \times \beta^{t(i)} \tag{5.5}$$

This memory is complemented by a problem specific 'Bother Increase Function', which is among the four problem specific functions which are required for the system to be used. We do not constrain this function, but note that its inputs (i.e. the expected net increase in utility produced by the interaction, the resource's BSF, and the resource's type) allow for the expression of a wide range of possible bother cost increase functions. For example, the Attentional State Factor (ASF; a measure of the extent to which the user able to pay attention to the system) and User Unwillingness Factor (UUF; a measure of the resource's interest in actually acquiescing to the request) criteria adopted for use in the bother increase function of Cohen et al. [10] can be roughly modeled by making UUF a function of the expected net increase in utility, and ASF a function of the agent's type and BSF.

The algorithm used by resources to update their bother model at each time step simply consists of discounting the resource's BSF by a factor of *beta*. It consists of one step:

```
1: SET $r.bother\_model.value \leftarrow r.bother\_model.value \times \beta$
```

As we shall see in the following subsection, the value itself is increased dynamically by the proxy agent every time it forwards a request to the resource itself. This is done instead of an aggregation at the end of a time step to allow the model significant leeway for adaptation within a step. If a desirable resource suddenly becomes available, it is not implausible that dozens of meritorious requisitions could appear within a single step. A proxy agent who waits until the end of a timestep to perform the resulting updates to its bother model will allow them all to reach the resource, a situation which is unlikely to be optimal.

Resource Proxy Agent Behaviors and Coordination

Resource proxy agents serve two important roles in the system. First, they maintain and update a model of the resource and its bother. Second, they respond to communication from other agents by reflecting on their knowledge of their resource and providing estimates of the changes in net utility which would result from various actions.

The first role is conducted according to the resource model outlined in the section above, but the second role is rather more complex. There are two types of queries that system agents can issue to resource proxy agents. First, system agents can confirm with resource proxy agents that asking a resource to take over processing of their task at the present time would produce a net increase in the overall utility of the allocation (see line 15 of algorithm 5.2). This behavior is shown in algorithm 5.7, and is essentially analogous to that of a type IV agent from Cohen et al.'s proposed system for managing bother costs [10].

The second type of query an agent can issue occurs during plan generation, implied by the passing of the function *compute_preemption_cost* to the TOC strategy generation method on line 5 of algorithm 5.3. This function computes the expected benefit of having resource r begin processing task t at a time τ. There are four possible situations to consider in this computation. First, we need to consider whether r is currently processing another resource. If they are not, then the expected utility of the request will be whatever t gains from its success, less any bother costs incurred by issuing the request in the first place. If r is currently processing someone else, we need to use the formula for $EU_{preempt}$ from equation 4.1, which describes the difference in expected utility if the preemption were allowed or denied respectively. Within these two broader categories, we also need to consider whether or not t already has a resource. If they do, then their increase in utility will be proportionate to the improvement in the expected time of completion for the task. If they do not, then it will be proportionate to that same improvement relative to a problem specific value returned when *processing_time* is called with the special type θ_r = 'useless'. Additionally, we need to project these beliefs forward in time so that if, for example, a task will finish before τ or a resource will become free by τ, the estimates are based on these expectations and not the present state. See algorithm 5.8 for a formal description of this process.

Algorithm 5.7 An algorithm describing the behaviors of a resource proxy agent when confirming the validity of a preemption request.

```
1: $allow\_transfer\_attempt?(proxy, task)$
2: **if** $compute\_preemption\_cost(proxy.owner, task, 0) \geq 0$ **then**
3:     **return** TRUE
4: **end if**
5: **return** FALSE
```

5.2.4 Task Behaviors

The final section describing the behavior on the system concerns the behavior of tasks. Task behaviors are comparatively simple in the abstracted version of the system. The *step_task* function actually has no effects. However, in a problem specific implementation, it can be redefined to produce more interesting behaviors. For example, tasks might cause new tasks to be added to the system when they are completed. As a consequence, the only task-specific function is *processing_complete*(*task*). In the generic system, this function returns TRUE if *task.processing_time_remaining* == 0, and FALSE otherwise. We note again however, that other definitions are both possible and interesting. For example, we might want to return boolean values in a probabilistic fashion to simulate uncertainty. Naturally, in a deployed, rather than simulated system, this function would require confirmation from a resource that the task really has been completed.

Algorithm 5.8 An algorithm describing the behaviors of a resource proxy agent when confirming the validity of a preemption request.

```
1: $compute\_preemption\_cost(r, task, tau)$
   {If the task's processing will be complete before we ask $r$ to start processing it, no improvement in EU is possible.}
2: if $exists?(task.current)$ and $task.processing\_time\_remaining \leq \tau$ then
3:    return 0
4: end if
   {Compute the expected processing times for $task$ under its current resource and $r$}
5: LET $pt_{r,task} \leftarrow processing\_time(r, task)$
6: if $exists?(task.current)$ then
7:    LET $pt_{old,task} \leftarrow task.processing\_time\_remaining - \tau$
8: else
9:    LET $pt_{old,task} \leftarrow processing\_time('useless', task)$
10: end if
11: if not($exists?(r.current)$) or $r.current.processing\_time\_remaining \leq \tau$ then
12:    return $EU(pt_{r,task}, pt_{old,task}, \theta_{task}, task.time\_waited)$
13: else
14:    LET $ptr = r.current.processing\_time\_remaining$
15:    return $EU(pt_{r,task}, pt_{old,task}, \theta_{task}, task.time\_waited) + r.current.agent.EU\_plan$
       $-EU(ptr, processing\_time(r, r.current) + pt_{r,task}, \theta_{r.current}, r.current.time\_waited)$
       $-EU(pt_{r,task} + ptr, pt_{old,task}, \theta_{task}, task.\_time\_waited)$
16: end if
```

5.3 Example

We now pause to consider once more the example from chapter 1, wherein failure to consider the opportunities for further resource acquisition following a preemption caused a suboptimal allocation, in which the life of a patient was needlessly lost. The system proposed above can address this issue and find the optimal allocation for the example.

To begin with, please refer again to figure 1.2. The figure shows three patients at a hospital, represented by green circles with the name of the patient inside them. Beneath the name of the patient is a deadline indicating where in the schedule (from left to right) the patient must appear in order to be treated without loss of life. For example, Bob has a deadline of 1, and so must end up in the leftmost spot of the schedule to avoid death, while Joe has a deadline of 2, and so can be placed in either of the two leftmost spots. Bob has arrived at the hospital last and is at the back of the treatment queue. In our original example, Bob asked Joe for his timeslot and was rejected, since Joe only considered what would happen if he had to move to the back of the queue. In our system, we would first assume that Joe and Charlie's agents already hold satisfactory resources and have made contingency plans (that is, they have generated TOC strategies describing what they would do if they could not keep their current resources). Bob's agent would first generate a TOC strategy at line 6 of algorithm 5.2, which would certainly begin with asking for Joe's timeslot. Bob's agent would then confirm with the proxy agent for timeslot 1 that its information is still accurate and that the request would produce a net gain in utility (lines 10 and 15). When this takes place, the proxy agent will ask Joe's agent about the value of its contingency plan as the second item in the sum of line 15 in algorithm 5.8. Joe's agent's contingency plan will first ask for the timeslot currently occupied by Charlie, which Joe's agent believes it is very likely to receive. Consequently, it will report an *EU_plan* value (the value of its contingency plan) equal to the value of continuing to hold its current timeslot, which completely cancels out the third term in the sum on line 15. Since these two terms cancel each other out, and the fourth term (the expected utility of allowing Bob to claim the timeslot only after Joe is done with it) is just the cost of Bob's death (which is negative), the returned value will contain only positive terms and be positive overall. Bob is thus allowed by the resource proxy agent to preempt the first timeslot, displacing Joe. Joe's agent then executes its contingency plan, and, through a similar chain of reasoning, is permitted to preempt Charlie's timeslot (since Charlie's contingency plan entails moving to timeslot 4, which is just as useful to him as timeslot 2). The optimal allocation is thus reached.

Note that in the discussion above, following Paulussen [51], we consider the timeslots as resources which, once allocated to patients, result in doctors being successfully assigned to carry out treatment. Our model as described in this chapter is designed to enable proxy agents (incorporating various user modelling elements) to directly represent humans (such as doctors) who would proceed with the desired actions. We return to this richer view of medical resource allocation in chapter 6.

5.4 Evaluations

This section describes the evaluation of a prototype of the system on a simple, abstracted, resource allocation task. The goal of these simulations is two fold. First, we aim to determine the relationship between values of the various system parameters and performance in terms of an abstracted 'cost' metric that measures the tendency of task to be completed in an efficient ordering, and thus, for resource assignment to be efficient. Second, we aim to determine the interaction effects of the parameters in consort. Please note that in this chapter we do not explicitly measure the value of including TOCs to address preemption compared to other approaches, instead focusing largely on the effect of the learning components. In chapter 6, this former issue is given greater attention within the context of an example medical application.

In our experiments, we use the following 'within subjects' design. First we generated 100 unique sets of tasks and resources using different random seeds. Then we ran the system with each of the parameterizations we wished to compare, with each of the 100 sets generated earlier as input. To determine whether any two parameterizations produced different performance, we used a paired t-test to check for a statistically significant difference in the mean difference between the two parameterizations across the 50 random problems. To avoid the increased false positive rate produced by multiple hypothesis testing, we used a threshold p-value of $0.05 \times \frac{2}{n \times (n-1)}$, where n was the total number of parameterizations in the system (this is a Bonferroni correction). For example, if we had 3 parameterizations that we wished to compare, and we wanted to accept a difference due to chance as a real effect with probability 0.05, , we would need a confidence level of $\frac{3}{60} = \frac{1}{20}$. Thus, each of the three tests should be performed at a confidence level of $\frac{1}{60}$.

A drawback of this experimental design is that it becomes quite hard to analyze the effects of parameter interactions, because the ability of our tests to detect differences will drop rapidly as the number of tests increases. If we have many parameters, each of which has many different values, then testing their effects together in a factorial design could require a very large number of repeated measures (probably more than 100). Thus, our experiments proceed in two steps. First, we analyze the effect of each parameter in isolation, holding all the others constant. This gives us some insight into the effects of many different parameter values. Then, using these insights, we design a single experiment that attempts to measure the interaction effects of the parameters using a few representative settings. This analysis is by no means comprehensive, but provides a reasonable characterization of the system's behavior, which could be used as a starting point for future practitioners.

Results are usually presented as plots showing, for each parameter set, the mean value of time taken or cost. Plots typically are accompanied by whiskers which denote a Bonferroni corrected 95% confidence interval based on a t-statistic for the mean. Thus, whenever the whiskers for two points do not overlap, there exists a statistically significant difference in their performance.

5.4.1 Experimental Problem Domain

The abstracted problem domain considered in subsequent experiments is formally defined as follows.

Task types are drawn with uniform probability from the set of integers between 1 and a user defined limit. Resources are assigned a random number between 1 and 10 for each possible agent type, with a resource's type corresponding to this list of 'skills'. The expected time taken for a resource to process a task is 11 timestpes, less the resource's skill at processing the task's type. For example, a resource with skill 6 at processing tasks of type 5, would take $11 - 6 = 5$ timesteps to process a task of type 5. This characterizes the *processing_time* function.

The $net\_EU(t_{new}, t_{old}, \theta_t, time\_waited)$ function is given by $(t_{old} - t_{new}) \times \theta_t$, that is the expected utility of finishing a task one timestep earlier is equal to a task's type.

The function $P_{stop}(net\_EU, BSF)$ is given by the average of two sigmoidal response functions:

$$P_{stop}(net\_EU, BSF) = 0.5 + \frac{1}{1.5(1 + e^{-net\_EU})} - \frac{1}{1.5(1 + e^{-BSF})}$$

with the result being capped between 0 and 1. This simulates resources which are eager to help when utility increases are available, but also disengage rapidly from the system when bothered.

Finally, we used the bother increase functions suggested by Cheng [5], so that requests expected to generate an improvement in net EU of more than 50 produced an increase in bother of 3/4ths the resource's BSF value, those producing a change of -50 or less an increase of $1.25 \times BSF$, and those producing a change between these extrema of min(BSF,1).

5.4.2 Experiment 1: The Effect of Plan Length

The first experiment measured the effect produced by changing the maximum plan length present in the system. Two hypotheses were considered. Hypothesis one was that, all else being equal, it is better to have a maximum plan length greater than 1 (i.e. that planning is better than not planning), justifying the inclusion of proper plans. Hypothesis two was that allowing agents to adapt the lengths of their plans over time produced an overall positive effect, justifying the inclusion of the agents' plan length learning models.

To test hypothesis one, we used the primary experimental design described at the beginning of this section. We used simulations with 100 resources, 1000 tasks, 500 possible task types, and 1000 agents. The maximum plan length an agent could use was varied so that $k \in [1, 16]$. Agent learning models were initialized so that plan lengths would start in the middle of the range $[1, k]$, at $\lfloor \frac{k}{2} \rfloor$.

The results of this process are summarized in figures 5.3 and 5.4, which show the impact of changes in k on the total cost and simulation time. Cost is measured as the priority of a task(from 1 to 500) multiplied by the time taken for the task to be completed in the simulation. Thus, a high cost could be indicative of either a small number of high priority tasks going unfinished for a longer time, or a large number of lower priority tasks. The simulation time metric is an indication of the total number of simulation timesteps taken before all tasks were processed. This measure usually correlates strongly with cost in practice, but a higher cost coupled with a lower simulation time can be indicative of a system which is prioritizing jobs incorrectly.

We find that more detailed plans tend to produce larger costs, but also paradoxically take much less time to resolve all tasks. A more detailed analysis of the results determined that, as the length of plans increases, agents tend to spend large amounts of time 'churning' at the start of a problem. During this initial churning period, little work is done, as agents learn about the environment and shuffle resource around. Eventually, usually after around 10 timesteps, the agents settle into an efficient allocation. Thereafter, work proceeds very quickly, leading to the lower completion times. Unfortunately, since we have made no effort to discount the costs of this learning process (which could in theory happen either offline or through a more careful selection of initial values for learning models), costs appear higher. We interpret this as an indication that longer term planning is of some value, but that practitioners must be careful to allow the system some time to settle before active use.

To test hypothesis two, we repeated exactly the experiment used to test hypothesis one, but with an ablated version of the system. In this ablated version, we produced a single change, namely, we added an extra statement in the *update_models* function described in

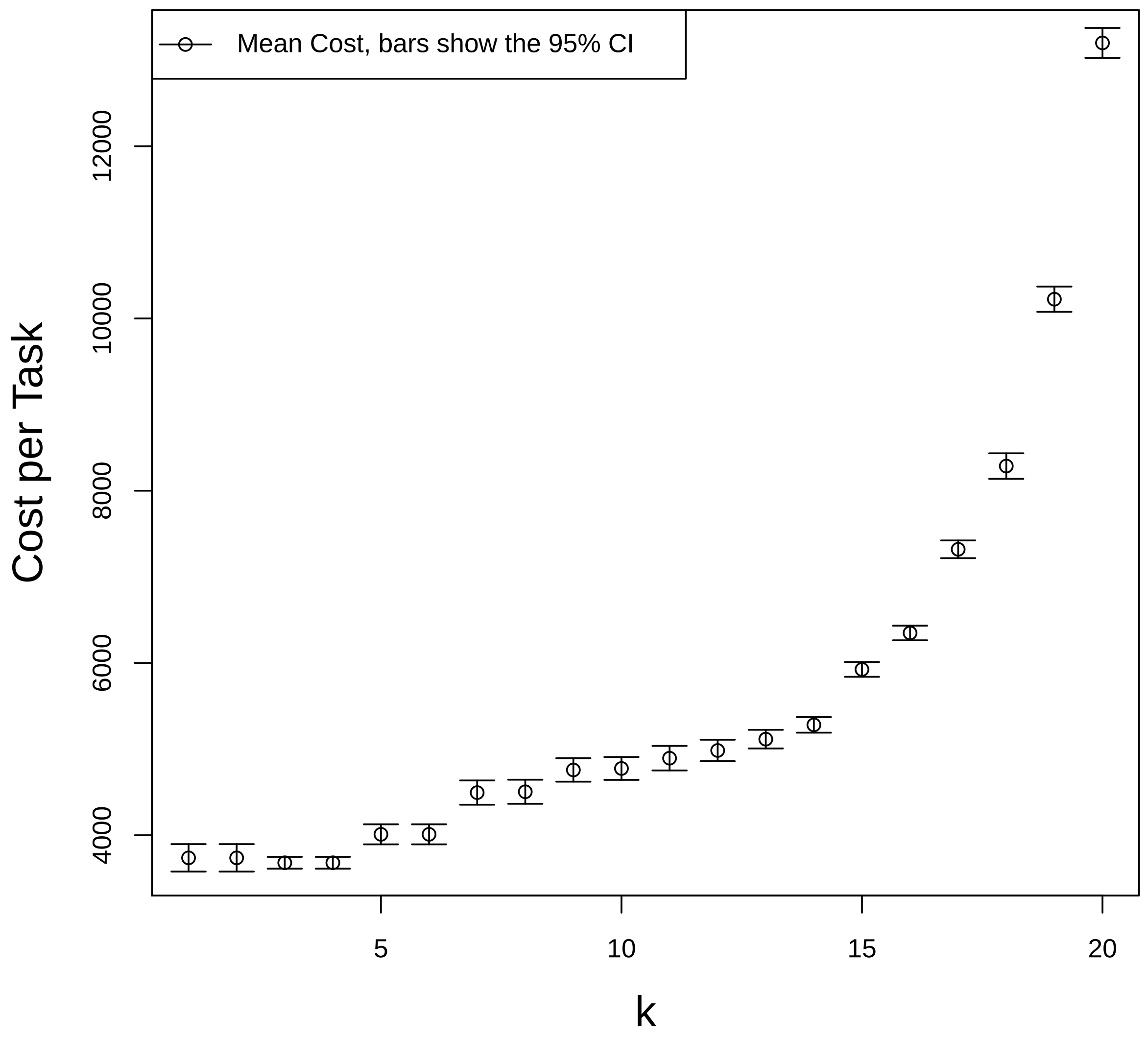


Figure 5.3: A plot showing the effect of increasing the lengths of plans on the total cost of finding a good resource allocation.

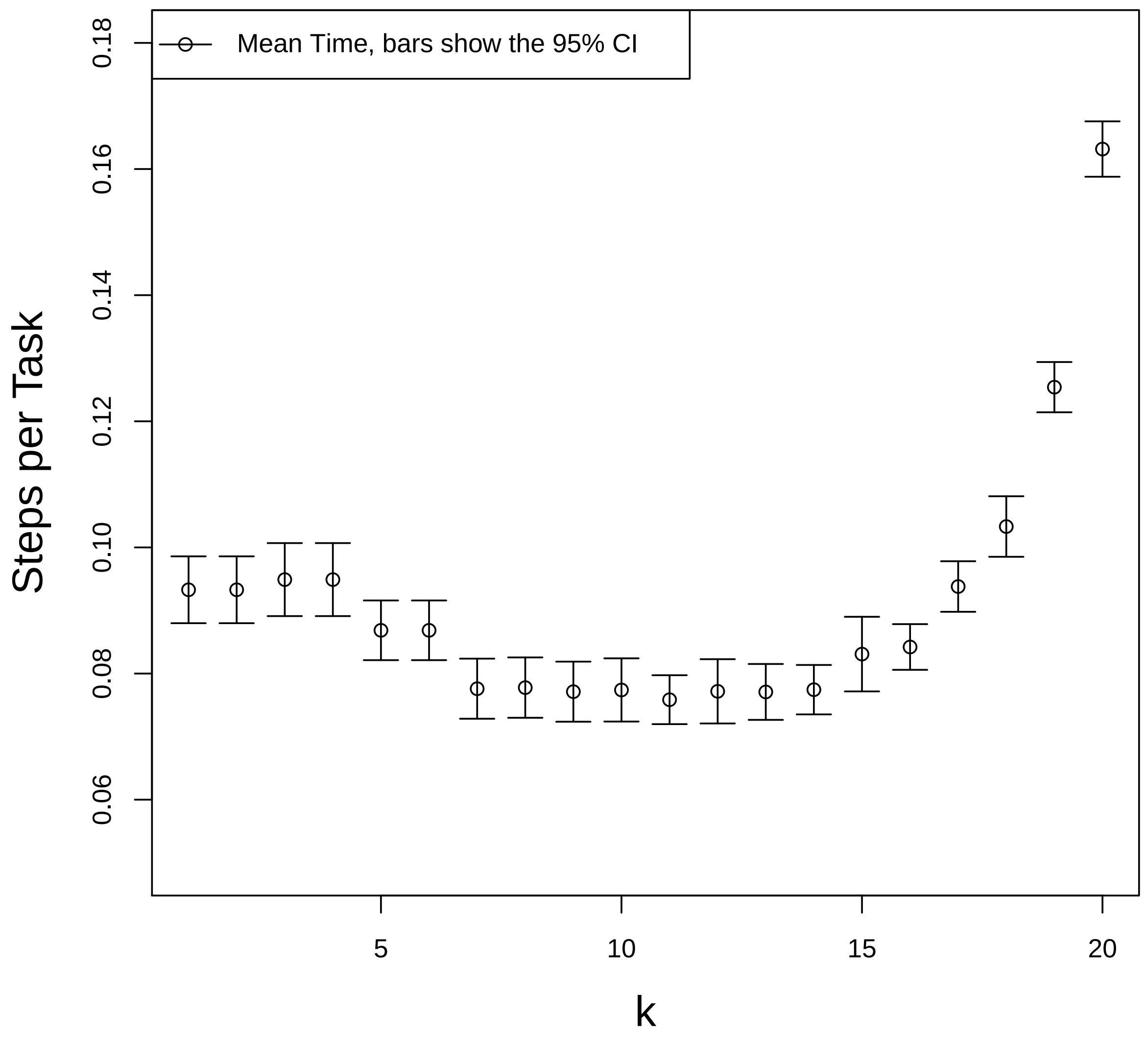


Figure 5.4: A plot showing the effect of increasing the lengths of plans on the time taken to complete all tasks.

algorithm 5.4, between lines 7 and 8, which set *a.plan_length_model.value* to zero. The result of this change was that agents always used plans of the same length (k).

We ran this ablated model on the same 100 problem initializations used to evaluate hypothesis 1, obtaining measurements of cost and time in the same fashion. We then subtracted the cost and time performance of the ablated model from that of the intact model. The resulting statistics have the property that values greater than 0 indicate the ablated system outperforming the original, while values less than 0 indicate the opposite. Unfortunately, the ablated system tended to produce wildly optimistic estimates of the costs of preemption, resulting in cyclic behaviors where agents churn resources back and forth without ever getting work done. As a consequence, simulations could only be completed for extremely small problem sizes with extremely low congestion, and were abandoned. We conclude a rudimentary learning component is a necessary requirement for the system to function.

5.4.3 Experiment 2: The Effects of Congestion

An additional important characterization of our system is the effect of congestion. We expect that our system will be more useful in environments with comparatively higher congestion, because preemption cycles are less likely to be an issue in environments where agents have less to gain from preemption. For example, in a system where all agents can have a resource that suits them well, our system is not very likely to offer an advantage.

To characterize the extent to which this is the case, we vary the number of tasks and agents in the system while holding all other parameters constant, with 100 resources, 500 agent task types, and a maximum plan length of 10. We always used the same number of agents as there were tasks, and denote this number as n. Using the primary experimental design described at the beginning of this section, we varied n between 100 (one resource per task) and 2000 (20 tasks per resource), in increments of 100. Like in experiment 1, we tracked both the total time taken for the system to clear all tasks, and the total cost accrued. Our findings are summarized by figures 5.5 and 5.6.

The data in this experiment exhibit a comparatively high degree of organization, so we have fit models to predict the effects of increased congestion on both cost and time. The cost model we fit is linear, and predicts an increase in the costs of *every* agent of 2.6 per additional agent in the system once congestion sets in (i.e. once there are more tasks than resources). This model is shown on figure 5.5 as a dashed red line. We compared this with a null model, based on the expected cost of treatment per task in a simple heuristic algorithm which makes random allocations and does not permit preemption (i.e. tasks hold resources until processing is complete). We let c_{task}, the cost per task, be the product of $E[c_{step}]$, the expected cost per agent per step, with λ, the expected time until a task is processed. In this simulation, $E[c_{step}] = \frac{500}{2} = 250$, since we had a uniform distribution over 500 types and an agent's cost per step is equal to its type number. λ can be further decomposed into half the number of agents to be processed times the mean processing time. Assigning a random agent to a random task in this simulation yields an expected processing time of $\frac{11}{2} = 5.5$, so $\lambda = \frac{5.5n}{2\times 100}$, and our final model for c_{task} under the assumption of random allocation is

$$c_{task} = E[c_{step}] \times \lambda = 250 \times \frac{5.5n}{200} = 6.875 \times n$$

Since our model's expected cost per task appears to grow at a rate of just $2.6 \times n$, we can draw two conclusions. First, our model is extremely resistant to the effects of congestion. If agents were wasting more time as congestion increased, we would expect to see a cost per task that was super-linear. Since we do not see this, we conclude that the new system is modelling the effects of congestion well, and is able to perform normally in environments with up to 20 tasks per resource. Second, we can conclude that the model is performing efficiently in areas of high congestion, based on its outperforming the random model. We note, however, that the costs of the learning period appear to dominate in our system until a congestion level of roughly 2 tasks per resource. Additionally, we note that our solution must be quite close to optimal performance in this domain. Although computation of increase in costs with congestion is non-trivial for an optimal allocation system, we note that, assuming we have 100 resources with the highest level in every skill, adding a new task to the system will still increase the expected cost per task by $1.25n$. In this case, mean processing time will be $\lambda = \frac{n}{2\times 100}$, since a resource with maximal skills will always take just one timestep to process any task. The expected cost per timestep remains the same at 250, so the total cost per task is $1.25n$. Since the existence of so many high value resources is highly unlikely, this serves only as a loose lower bound, but still suggests that our system is within a factor of 2 of the optimal allocation algorithm, which must experience costs strictly greater than $1.25n$ per task, while our algorithm experiences an increase of $2.6n$.

The model for the effect of congestion on timesteps is somewhat more complex. The data appear to follow a rational expression of the form $\frac{\alpha}{n} + \beta$ for some parameters α and β. This is consistent with each task taking a constant amount of work (β), and amortising some fixed cost α. Taking the intercept value in our linear fit for the effects of congestion on cost (2200), and dividing it by the expected cost per agent (250) suggested an α value of roughly 8.8, implying that agents spend, on average, 8.8 time steps learning before they can begin effective allocation. We find a good fit with values $\alpha = 8.8$ and $\beta = 0.046$, and propose that α and β correspond to the mean learning period and the processing rate for the system post-learning respectively. This would indicate an average wait time of $\frac{4.6n}{2|R|}$, as opposed to $\frac{5.5n}{2|R|}$ in a random allocation, meaning that the system is performing quite efficiently. The model also provides further evidence of the system's resilience to congestion, since it does not grow with n.

As in Experiment 1 above, we were also interested in the effects of our learning model on congestion, however, we found it impossible to conduct reasonable experiments without the preemption learning model because, at any reasonable value of k, the maximum plan depth, failure to discount the value of plans results in extremely optimistic (much too low) estimates of the costs of preemption. Consequently, agents end up swapping around resources indefinitely without actually doing any work. We conclude that this component is an essential part of the system.

5.4.4 Experiment 3: Combined Effects of Congestion and Planning

Our final experiment examined the interaction between the level of congestion and the extent of planning. We did not anticipate any particular type of interaction effect, only that an effect might exist. For example, it might be the case that, when the system is more congested, more detailed plans are more useful, or it might be the case that the learning component of the system eliminates any such difference.

In this experiment, we used 4 values for plan lengths ($k \in \{1, 4, 8, 16\}$) and four degrees of congestion ($n \in \{100, 400, 800, 1600\}$). All 16 combinations of these two parameter sets were run, with the remaining parameters being set to 100 resources, and 500 agent types. As in experiments 1 and 2, we tracked both the total time taken for the system to clear all tasks, and the total cost accrued. The multi-dimensional nature of the results do not lend themselves well to a visual representation, so we have spread the results across several figures. As this process can make it difficult to visually infer the results, we also include a more detailed discussion both of key findings and the techniques used to measure them.

Our first major finding is that there is a significant interaction between the level of congestion and the optimal maximum plan length.

We determined this by performing ANOVA on the 1600 data points obtained from our experiments on both the cost and timestep measures. ANOVA is a specialized rephrasing of linear regression intended for use in factorial experimental designs like the ones we used here [24]. Essentially, it determines how much of an effect each parameter in an experiment

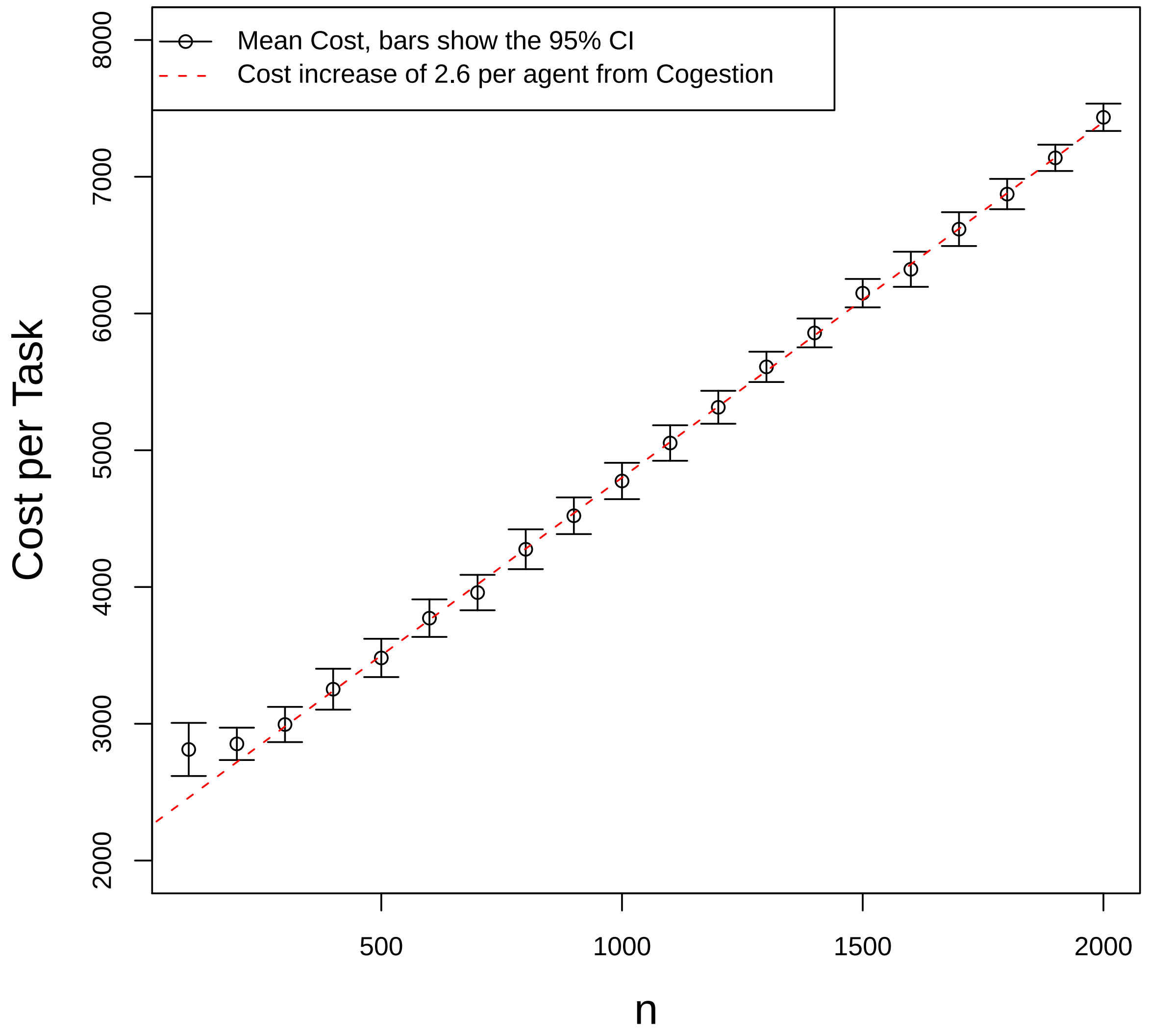


Figure 5.5: A plot showing the effect of increasing resource congestion on the total cost of finding a good resource allocation. The dashed red line shows a predictive model.

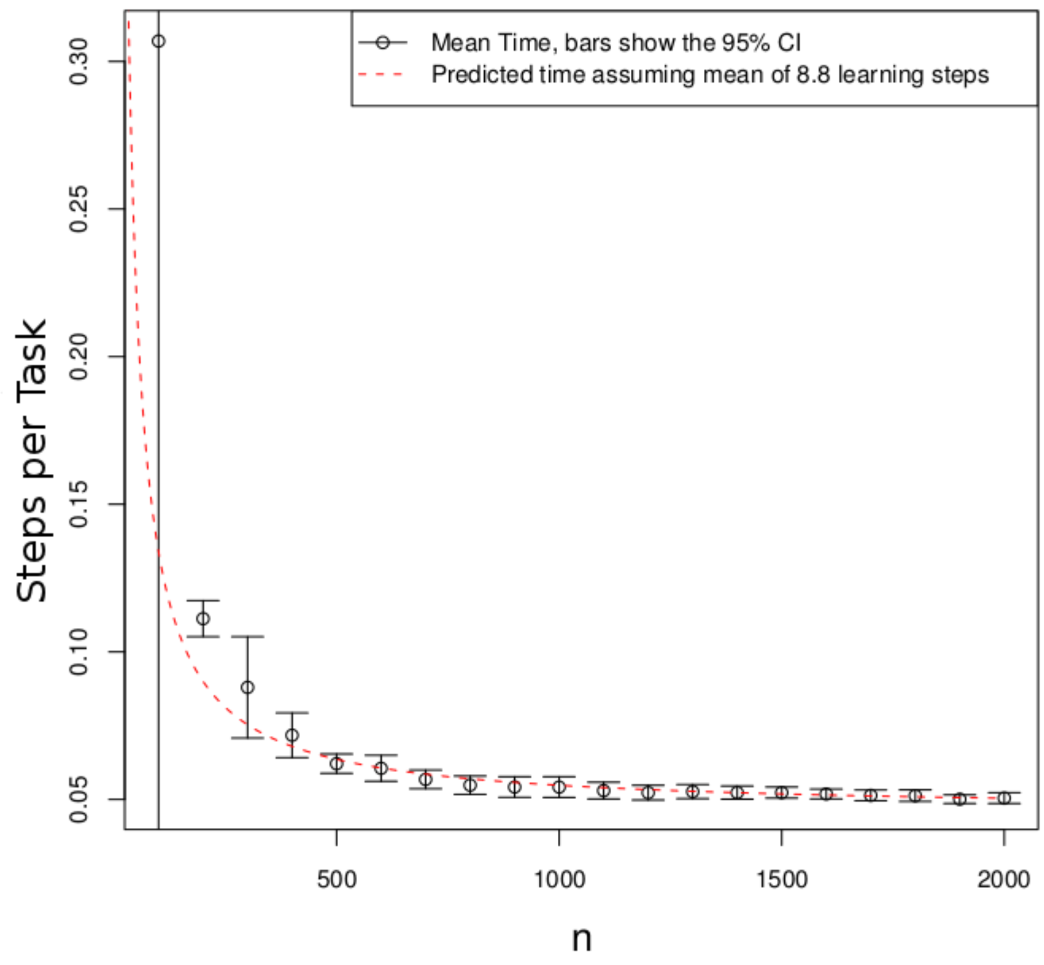


Figure 5.6: A plot showing the effect of increasing resource congestion on the time taken to complete all tasks. The dashed red line shows a predictive model.

is responsible for, while also estimating the effects due to nonlinear interactions. We used the analysis of variance function from the stats package in the statistical language R [75] to obtain ANOVA estimates of the effects due to each parameter setting and due to the interaction between them. In this context, a significant interaction term is indicative of different plan lengths exacerbating or mitigating the effects due to increased or decreased congestion. **For practitioners** this means that changes in the congestion of the system can produce non-linear changes on the effects of choosing a particular value of the parameter k. Figure 5.7 depicts this difficulty. Although a value of 8 is often better than the alternatives, a value of 4 is slightly better in a congestion free environment, and a value of 16 may be preferred in the 8:1 environment. Consequently, in light of both this and the coefficients extracted from our ANOVA, we can recommend that, if scenarios have more congestion, practitioners should prefer longer plans. In contrast, when there is not much congestion, practitioners should prefer shorter plans.

In general, the interaction effect appears to be small (on the order of 1-3% of system time and costs). However, a general trend visible in the coefficients of the regression model underlying our ANOVA is that more congested environments provide non-linear benefits to having larger plan lengths. We postulate that this is the result of highly congested environments producing fairly 'stable' allocations (which benefit long term planners). For example, if all resources are busy every time you make a plan, and they remain busy (even if their owners change) for the duration of the plan, the detailed plan was a reasonable thing to construct. For the parameter values examined in this experiment, this relative effect is still dominated by the absolute benefits of having mid-length plans discussed in experiment 1 above, but it is not unreasonable to expect that, in even more radically congested systems, the effect might come to dominate, leading to a preference for longer term planning.

5.4.5 Discussion

We have shown that the system described in this chapter is basically sound using a simulated scheduling task. The system is highly robust to congestion (and thus, to preemption cycles), one of the primary goals of its construction. Its learning components are vital for functionality, but picking poor initial values or learning parameters can hold a significant cost. Nonetheless, once this cost has been accounted for the system's performance appears strong.

This suggests that the primary avenue of future development for the system should be the introduction of more sophisticated learning algorithms and/or research into optimal

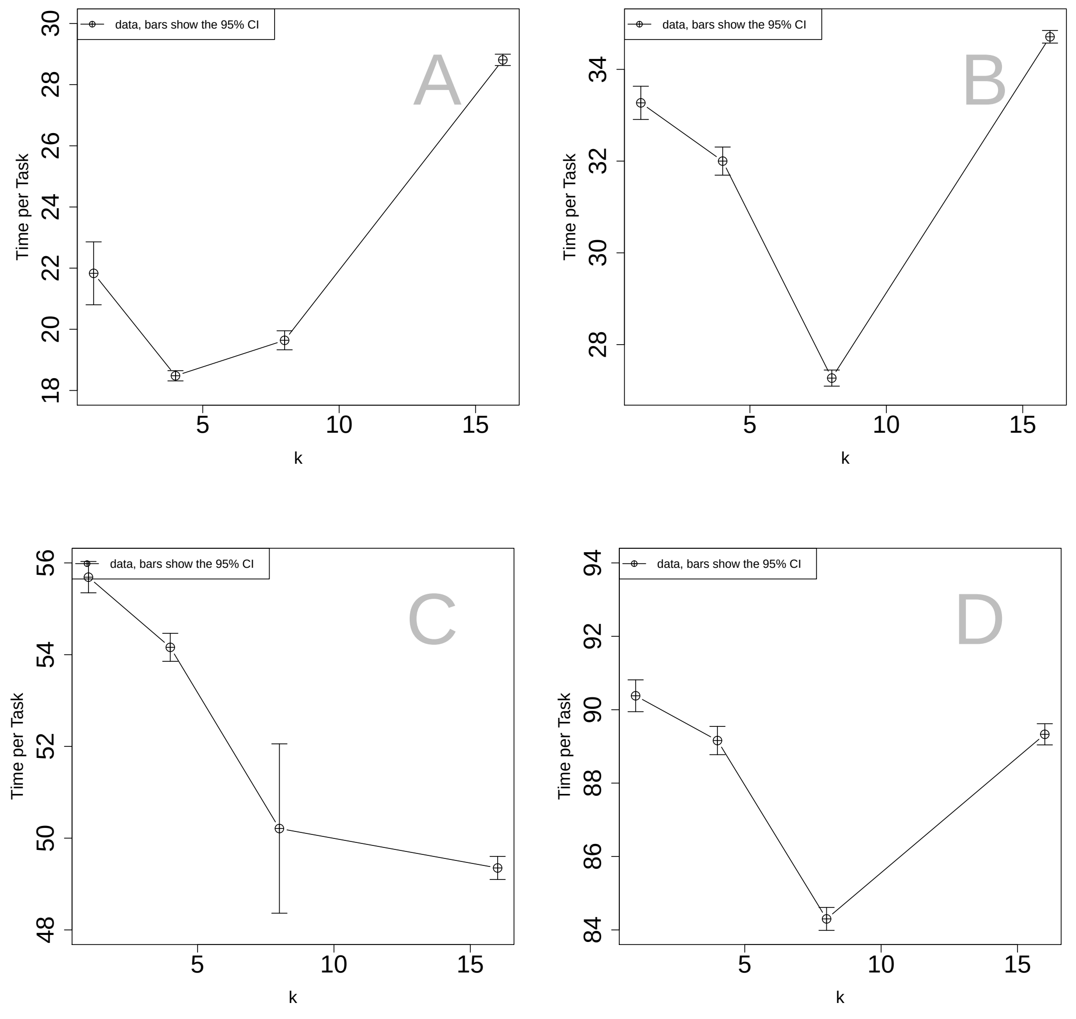


Figure 5.7: Plots showing the difficulty of picking an optimal value of k as congestion levels change. Each plot shows the relationship between k and the total number of timesteps taken for a different level of congestion (Tasks:Resources). (A) shows 1:1, (B) shows 4:1, (C) shows 8:1, and (D) shows 16:1

learning parameters and initializations. We expect that the incorporation of even a simple gradient-based learning algorithm could virtually eliminate the system's initial 'learning period', and could also lessen the system's vulnerability to sudden changes in congestion levels.

We have also confirmed the relationships between congestion and churn. In particular, we found that larger degrees of congestion increased churn, causing optimal plan lengths to be longer (which is intuitively expected). In an environment with very low congestion, agents will generally be able to acquire any resource they please, and hold it for as long as they desire. Most of the simulation will thus be spent in a 'steady state', with very occasional interruptions. In this environment, it is reasonable for agents to build elaborate contingency plans, since the problem is likely to remain stable long enough for such plans to be executed in full. In contrast, a high congestion environment exhibits the opposite characteristics. Even if an agent can acquire a resource, it is likely the resource will be taken from them (as modeled by the congestion learning process, which discounts the expected value of acquiring the resource in the first place). Naturally, if this is true, the problem can be expected to change again momentarily. If many situations like this are happening at once, churn will be high, and agents should make shorter plans.

Chapter 6

Applications to Emergency Medicine

In this chapter, we apply our methodology to the motivating medical application of assigning doctors to patients in hospitals during 'mass casualty incidents' and demonstrate noticeable improvements in performance (generating far fewer problem patients) than competing approaches that do not effectively model the opportunity costs that arise with preemption and the challenges of cyclical dependencies.

The chapter serves to showcase an application of an early version of our system (a version which predates the model in chapter 5) which utilized TOC strategies to resolve the problem of preemption cycles effectively. Unfortunately, the bother models utilized make this problem non-Markovian, and so the non-Markovian dynamic programming algorithm from chapter 3 (algorithm 3.3) was used, which serves to illustrate how practitioners may decide which algorithm from chapter 3 to employ. This algorithm is quite slow, and the resulting limits on experiment sizes ultimately motivated the inclusion of learning algorithms in the full system from chapter 5, allowing for the much more efficient Markovian dynamic programming algorithm to be used even in non-Markovian scenarios. The high computational cost of the algorithm used in this chapter has naturally limited the problem sizes which can be usefully examined, and thus smaller input sizes are used in this chapter than in chapter 5 (i.e. fewer resources or patients per problem instance, smaller values of k).

6.1 Introduction

Recent work in the application of multiagent systems to appointment allocation problems in the medical domain [50] has produced promising results through the use of a market-based coordination mechanism. The MedPAge approach adopted by Paulussen et al. [51] however, falls short in scenarios where preemption of tasks is an issue, because it utilizes a worst case estimate of the opportunity cost of a resource preemption.

In this chapter, we demonstrate how a specific type of high-congestion medical scenario could be modeled using an earlier version of the framework described in chapter 5. We then demonstrate through simulations that the new opportunity cost model can better balance the requirements of patients during situations where very large numbers of patients are present at once, as in a *mass casualty incident*, and so can reduce the frequency of 'problem patients' (i.e. preventable deteriorations of patients into critical status) which tend to take far more money and time to treat than typical patients.

Thus, the chapter serves both as a case study for modelling a problem using our proposed system, and as a demonstration of the system's usefulness when compared with an alternative approach.

6.2 Background

6.2.1 Medical Scheduling and Resource Allocation

Scheduling and resource allocation problems are tightly integrated with modern medicine, which often presents singular challenges due to the highly dynamic nature of the problem domain. Historically, many hospital scheduling problems have been addressed using techniques from optimization (especially integer linear programming [13]), and heuristic guided search [54][55]. Hospital scheduling problems which are not highly time dependent are often phrased as integer linear programs (ILPs) and solved directly, producing optimal solutions at the cost of super-polynomial worst-case run times. This approach has been successfully applied to shift scheduling for nurses[34], allocation of hospital resources for major projects [38], and long term scheduling of teams to operating theaters for routine surgeries[4]. A drawback of these techniques is the very high worst-case run times, which may have difficulty keeping up with rapidly evolving situations. If, for example, it takes us 2 hours to make a schedule, but 20% of our patients have recovered and been replaced by new and different patients in the mean time, our schedule is unlikely to be optimal. A secondary disadvantage is the lack of clear information about the reasoning behind the resulting schedule - a schedule generated using optimization techniques cannot easily offer a reasoned justification for any particular subcomponent contained within it or any particular change in the schedule, beyond the tautological 'because it improves efficiency'. This can create problems when, as in the medical world, such schedules are intended to be used by humans rather than by software agents. For example, a schedule which delays treatment of a patient in stable, but unpleasant, condition to treat one who is apparently fine but has a small probability of developing a life threating prognosis, may appear incorrect to a staff person and so may be disregarded. Heuristic techniques are also widely used, especially for nurse scheduling and for solving medical location-allocation tasks (e.g. [54][55]. These methods generally suffer from the same drawbacks as optimization techniques, again making them poorly suited to online scheduling tasks involving human subjects. Finally, Friha et al [27] and Hannebauer [28] proposed early distributed approaches similar to multi-agent solutions, which were grounded in constrained optimization, but which are not concerned with dynamic problems where preemption can be an issue.

Recently, partially and fully distributed agent based approaches have begun to appear in the hospital scheduling literature. One of the principal reasons for adopting these approaches is the inherently distributed nature of the hospital scheduling problem: each department within a hospital may set its own schedule independent of the schedules used by other units, which can create inharmonious global schedules[15]. Decker and Li applied their Generalized Partial Global Planning [14] algorithm to hospital scheduling to improve throughput and decrease the duration of patient stays [15], but acknowledged that their approach did not generate optimal schedules (though this was justified by noting that optimal plans for such dynamic environments rarely produce substantial improvements). Vermeulen et al. proposed a different approach through the utilization of automated Pareto efficient appointment time swaps between different patient agents [78]. This system had the advantage of being incentive compatible for patients, and of being fully distributed, but limiting scheduling changes to voluntary exchanges and incremental improvements can result in sub-optimal scheduling when the incentives of individual patients do not align with the goals of the hospital itself. Paulussen et al. [51] utilized a *market based* coordination mechanism called MedPAge for the problem of patient scheduling, in which the agents representing patients attempt to obtain appointments that maximize improvement in their patient's 'years of well being'. This was accomplished through negotiated timeslot swaps, wherein current holders of a slot would relinquish it if they expected that the adverse effects on their patient would be less than the positive effects of allowing the swap. The mechanism is market based in the sense that the preempting agent specifies an amount it would be willing to pay in order to facilitate the swap, and the present holder of the timeslot will relinquish it only if the amount exceeds its expected losses.

The model presented in this chapter offers a notable advantage over other approaches in that it allows for preemption of resources by agents which need them, and for determining when preemption should be allowed. As discussed in earlier chapters, the MedPAge system computes only a first-order approximation of the true costs of preemption. In this approximation, the present holder of a resource may consider the costs of preempting other timeslots itself when computing its expected losses, but agents holding those timeslots may only consider the outcome of moving to an *empty* slot when considering the preemption costs *they* would incur. This amounts to a worst case estimation of costs of these secondary preemptions, which can overestimate costs and prevent needed preemptions. Further, it imposes a *concurrency constraint* on the system, requiring agents to make preemption requests one at a time, and makes no previsions for the preemption of resources that are presently in use. In contrast, our system possesses the capacity for full distribution, allowing multiple agents to issue preemption requests simultaneously [51].

6.2.2 Mass Casualty Incidents

A mass casualty incident (MCI) occurs when a large number of patients simultaneously and unexpectedly suffer (potentially) critical injuries in the same time and place [47]. They are distinct from large scale emergencies such as floods insofar as such events are typically anticipated by communities in which they occur, and plans for mitigating their effects already exist. MCIs are widely studied, especially in Israel, which see a large number of bombings target civilians each year [22]. In regions where MCI are common, first responders rapidly reach the site of an attack and transport patients to a nearby hospital [31]. Crucially, patients are overwhelmingly likely to be transported to the *nearest* hospital, rather than the hospital best equipped to care for a large number of serious patients (the associated odds ratio has been measured at nearly 250) [31]. Consequently, MCI represent scenarios where a large number of critical patients may unexpectedly arrive at a hospital simultaneously, and so may pose problems for existing scheduling systems.

6.3 Model

To solve the cyclic dependency problem, we propose the incorporation of TOC strategies into a medical scheduling system, to provide better performance in MCI. The cyclic dependency problem is expected to become more pronounced as the number of patients present increases. Consequently, MCI are expected to serve as an especially strong example of improvements in performance through careful solving of the cyclic dependency problem, although the techniques we use are general purpose and could be applied to any multiagent scheduling or resource allocation problem involving preemption.

To phrase an MCI as a resource allocation problem, we need to define precisely how:

- Patients should be mapped to tasks.
- Doctors and other medical staff should be mapped to resources.
- The four **problem specific functions** from chapter 5.

We begin by mapping patients to tasks. In our medical setting, a task's type is represented by a condition, which defines a change in expected health state over time. For example, a patient might have condition 'broken leg', or condition 'shrapnel wound'. Naturally, these conditions can be made more or less detailed. For example, we might want to differentiate between 'elderly with broken leg' and 'young with broken leg', though in this work we do not consider such detailed differentiation.

Patients arrive at the hospital with a particular health state H (which, following [50] can be converted directly to a scalar measure of utility), representing the severity of their condition. This allows us to model the differences between two patients with the same condition, but different levels of progression. For example, a patient whose leg broke half an hour ago may well be fine to wait for some time before having the leg seen to. In contrast, a patient with the same broken leg condition who was trapped in the wilderness for several days prior to receiving treatment may suffer much more during the same waiting period. This tells us that the problem specific function *EU*, which describes the expected utility costs of making a particular task wait longer or short amounts of time for treatment should be dependent on a patient's health state, and that health states should deteriorate as time passes.

To model this, we make *EU* dependent on the length of time a patient has waited. There are many methods of doing this, but we adopt the model that for any given type, EU is given by

$$EU = e^{c(\theta_t)*(time\_waited+old\_time-new\_time)}$$

with some condition specific co-efficient $c(\theta_t)$ changing the rate of growth for the exponential. A patient's initial health state can then be represented by mapping it onto a non-zero *time_waited* value for a new task entering the system. In this work, we will assume that agents have access to the heath state data of their own patients.

Doctors and other resources are modeled, as in the simulations of chapter 5, as a set of skills which define their type. Processing times are a function of the skills doctors possess. For example, an expert surgeon might be modeled with high skill in treating wounds, while an intern might have much lower skills. Processing time is defined as a function of the type of a task and this skill level, but also of the patient's health state *H* (and thus, of *time_waited*), which requires a small modification of the framework from chapter 5. In particular, a doctor's skill reduces a patient's effective *time_waited* during each timestep that the doctor was treating the patient. For example, a master surgeon might reduce the effective *time_waited* of a wounded patient by 5 steps for each timestep of care, while the intern might merely slow the progression by reducing the *time_waited* by half a timestep.

In our model, a patient is declared 'cured' and the associated task complete if their health state ever drops below zero (i.e. the effective wait time experienced by the patient is negative). Similarly, a patient is declared a 'problem patient' (one at increased risk of death and with very high medical costs) if their health state is allowed to deteriorate above 200. The goal of the simulation is to produce the smallest number of problem patients possible.

In the interest of a more problem specific vocabulary, we define an instance of this doctor allocation problem as an eight tuple:

$$\{P, D, C, has, skills, initCrit, \frac{dCrit}{dt}, treat\}$$

where:

- P is a set of patients.
- D is a set of doctors.
- C is a set of conditions.
- $has : P \rightarrow C$ is a function mapping patients to conditions (e.g. $has(Bob) = Cancer$ describes patient Bob having condition Cancer).
- $skills : D \times C \rightarrow \mathbb{R}^n$ is a function mapping pairs of doctors and conditions to skill levels.
- $initCrit : P \rightarrow \mathbb{R}$ is a function mapping patients to their initial health states (criticalities).
- $\frac{dCrit}{dt} : C \times \mathbb{R} \rightarrow \mathbb{R}$ is a function of conditions and present health states that describes the change in the patient's future health state (i.e. the derivative).
- $treat : D \times C \rightarrow \mathbb{R}$ describes the relative change in $\frac{dCrit}{dt}$ produced by a doctor treating a patient with a particular condition.

We can then rewrite EU as

$$EU(\theta_t, t_{old}, t_{new}, t\_wait) = \int_{t_{new}}^{t_{old}} \frac{dH}{dt}(has(\theta_t), t, t\_wait)dt$$

6.3.1 User Modeling

In the case of *human* medical resources, we also incorporate techniques from user modeling to estimate the probability of response to a preemption. For example, a doctor making the rounds might be called from his current patient into surgery. If a system makes many preemption requests, especially requests which do not account for the severity of the situation currently being addressed by the medical professional, the professional may disengage from the system. Similarly, if the professional is involved in a sensitive task (e.g. surgery), they may not acknowledge interruptions by the system. Following Cohen, Cheng, Jung and Fleming [35][10], for each user, we adopt a user-specific exponentially decaying model of bother, where requests for direct preemption contribute to a human resource's bother, and the contribution is proportionate to the importance and relevance of the request. Relevance is estimated using a $User\_Unwillingness\_Factor$ (UUF) and an $Attentional\_State\_Factor$ (ASF), problem specific values which are based respectively on the user's ability or willingness to do the task in question and the user's attention level[5]. For example, a doctor who is presently doing routine work in a clinic would probably have very low UUF if asked to save an emergency patient's life instead. Once in surgery, the doctor would have a very low ASF from the system's prospective because she is unlikely to respond to requests while engaged in that task.

The base cost of initiating a new TOC operation in this model is given by $Init = UUF \times ASF \times TOC\_Base\_Bother\_Cost$ [25], where the last term refers to the inate difficulty of the task. Typically this is low when querying a user for their preferences and high when asking a user to make a complex decision, but the precise values are problem dependent. The ultimate bother cost of a TOC operation is given by $BC = Init + BC\_Inc\_Fn(BSF, UUF)$. The Bother Increase Function BC_Inc_Fn [25]describes the rate at which bother increases as a function of the user's present perception of how often the system is bothering them, and how frivolous the task appears to the user. Typically this is a function of the form BSF^{UUF}, where BSF is an exponentially weighted moving average of the bother from previous interactions (I), given as

$$BSF = \sum_I BC_I \times \beta^{t(I)}$$

and $0 < \beta < 1$[1].

For details on how we propose modelling the UUF and ASF of a resource in this medical application, see appendix A.

[1]In this chapter's experiments, following Cheng [5] and Jung [35] we assumed a constant β of 0.9 for all doctors.

6.3.2 Negotiation

The core of our new framework lies in its negotiation protocol. We use a protocol similar to Paulussen et al. [50], but with several notable differences. First, our protocol places no restrictions on the number of agents who can act at once. That is, there is no concurrency restriction in our protocol. Second, our protocol allows agents to interact with human actors directly, as well as with other agents. This is useful insofar as the system can be proactive in gathering information, or attempting direct preemptions (which might be required in MCI). The underlying bother model mentioned above helps ensure that the system makes only reasonable requests of human users. Third, our system allows for probabilistic response times as a natural consequence of supporting interaction with humans. Instead of receiving an immediate response, agents in our system expect some delay (following a known distribution), which might be caused by network latency, damage to the hospital, or prolonged reasoning (by humans or machines). Finally, our system does not make use of virtual currency. Instead, our agents simply relinquish resources to those with greater need without charge. We do this for two reasons. First, in an MCI, a market based mechanism can funnel large amounts of money to relatively unharmed patients who simply arrive first, and so who receive high priority initially. Second, we believe a fully cooperative agent system is incentive compatible in many cases because it produces positive externalities which can offset individual disutilities. For example, in MCI a patient would be content to know that the system places a high priority on the seriously injured, even if they are not seriously injured themselves. The basic negotiation process is presented below as part of the behavior of a patient agent throughout:

1. Let A be an agent representing a patient with health state H_A, and condition C.
2. Let P be a plan of action for A which specifies the optimal (or near optimal) order for A to request medical appointments (see Preemption Cost Estimation below).
3. Let p be the appointment time suggested for acquisition at the current time by P, and t_{old} be the present appointment time for A with the specified resource.
4. If p is unowned, acquire it.
5. Otherwise, if p is owned by another agent B, compare the gain made by A preempting p against the TOC-based **preemption cost estimate** produced by B.[2]

[2] This cost is simply the value B expects if it continues to hold p, less the expected benefit of executing B's TOC-based contingency plan.

6. If the gain exceeds the cost, A takes p, and B accepts the closest open appointment time as a temporary location, and now searches for a new appointment time based on its own plan of action (i.e. its TOC strategy).

7. If p is the last step in P, generate a new plan.

8. Otherwise, remove p from P, and GOTO step 3.

6.3.3 Preemption Cost Estimation

Our framework ties the costs of preemption to the generation of the plans of action mentioned above. In essence, a plan of action is a TOC strategy which specifies the sequence of TOC operations which is expected to lead to the optimal decision for the appointment time of the agent's patient. At each time step, an agent processes any responses received since the last time step, and then executes any TOC operations required by its plan. Whenever an agent acquires an appointment time, it computes a new TOC strategy relative to said resource. That is, it builds a TOC strategy describing a sequence of actions that would be taken if its current resource were suddenly unavailable. When another agent attempts to preempt the agent's presently held appointment time, or inquires about the present value of the appointment to the agent for use in construction of a plan of its own, the agent reports the expected utility of keeping its current timeslot, less the expected gain from executing the plan. The plan may also be periodically regenerated to avoid having it become out of date.

Pre-computing contingency plans in this fashion allows agents to respond to requests for preemption costs estimates without the need to query other agents at the time of the request. This resolves the cyclic dependency problem by providing a reasoned estimate of the true preemption cost without allowing cycles to form.

6.4 Experiment

We implemented a version of this model for an MCI scenario, and compared it with two ablated models which used myopic worst-case bounds to estimate preemption costs, following Paulussen et al. [51]. Our first ablated model estimated the cost of preemption by assuming that patients would go untreated for a large period of time following the preemption, and so greatly limited preemptions. This model is highly analogous to the estimates used in the MediPage system [51]. Our second model used a single step lookahead, estimating costs in terms of a plan of length 2, and was able to make more preemptions, representing a still simple, but somewhat more clever variation of MediPage. The comparison systems were otherwise identical to our proposed framework, including the same bother cost model, and the same negotiation procedure. Note that, although similar to the system of Paulussen et al., our ablated models are not identical. They do not involve the exchange of currency for instance, while they do incorporate more detailed user models. Although this is not a direct comparison, we believe it to be illustrative of the advantages of using longer TOC strategies.

Our simplified scenario featured four abstracted conditions, Routine, Minor, Major and Emergency. These conditions produced a value for $c(\theta_t)$ proportionate to 0.04, 0.08, 0.12 and 0.16 respectively. Thus, for example, a patient with a Routine condition and a criticality of 50 would have a criticality of $50 * 0.04 + 50 = 52$ if left untreated for a single time step. Medical resources were generated with uniformly random efficiencies for treating each condition. Efficiencies were drawn uniformly from the range $[0 - 20]$, and functioned as subtracted $\frac{EU(0,1,\theta_t,t.time\_waited)}{efficiency}$ from a patient's effective time waited during each treatment timestep. For example, a doctor with an efficiency of 20 for the Emergency condition, treating a patient with the Emergency condition would produce a change in health state of $\frac{dH}{dt} = 0.16 * H - 20$.

We limited TOC strategies to a fixed depth of 5 time steps ahead. We utilized the default values for the bother cost model suggested by [10] for simplicity. ASF and UUF were generated as a function of health state of a medical resource's current patient (if any) and the expected utility gain of the preemption respectively.

Our MCI scenario starts with a small number of patients at the hospital (drawn with uniform probability from the range $(0 - 15)$, all with routine conditions. The MCI is simulated by generating a large number of patients (the exact number is a simulation parameter, see below). Each patient was assigned one of the four conditions with uniform probability, and an initial health state drawn from a simulation specific distribution. Patients were then assigned an arrival time at the hospital, drawn with uniform probability from the range of $(0 - 20)$ time steps. At the time of arrival, each patient was assigned a new patient agent, which generated a new plan of action. We assumed that the hospital had sufficient capacity to hold all patients who arrived there.

Three parameters were varied in our simulations. First, the number of medical resources was varied between 5, 10 and 15. Small numbers were used to facilitate efficient computation on the limited computational resources. Second, the number of patients produced by the MCI was varied between 25, 50, 100, 200 and 400. Finally, the distribution of initial patient health states was varied between $U(0, 50)$, $U(0, 100)$, and $U(50, 100)$, where $U(x, y)$

is a uniform distribution over (x, y). Thus, for example, $U(50, 100)$ indicated that patients arriving at the hospital would have comparatively high initial health states, corresponding to a need for immediate treatment. In contrast, patients drawn from $U(0, 50)$ could wait some time for treatment before their health state had deteriorate substantially. In the first experiment, we utilized a single parameterization with 400 patients, 15 doctors and low initial patient health states. In the second experiment, we conducted runs on every combination of values for the three parameters. In each case, we ran both the new model and the ablated comparison model on the same 100 random seeds, resulting in 100 unique initial conditions. A patient who reached a health state of 200 was ruled a problem patient, and removed from the simulation in much the same way as a cured patient to insure the eventual end of the simulation. The simulations took approximately 100 hours to complete on the available hardware.

6.5 Results

Our results are summarized in figures 6.1 to 6.4. Each plot depicts the mean difference between the performance of our method and the ablated comparison model in terms of the percentage of patients who became 'problem patients' (%PP) during the course of the simulation, or the percentage of patients saved (from becoming problem patients) by using the new system (%PS). Tests of significance were done using parametric measures if the data were consistent with a normal distribution, and using non-parametric measures otherwise, and Bonferroni corrections were used where applicable. All plots were created and all tests of significance were performed, using R [75].

The results comparing our model with an ablated version incapable of preemption are shown in figure 6.1. The figure depicts three violin plots, which show three pieces of information each. First, the median value is shown with a white dot. The black boxes and bars show a box plot of the data, with the top whiskers, top of the black box, median, and bottom of the black box showing the four quartiles of the data. The grey 'violin' shapes surrounding the box plots show a smoothed density estimate for the distribution, which is reflected on either side of the box plot. This last piece of information can be useful in determining, for example, whether the data were bimodal, using only visual inspection. So, for example, the center violin plot tells us that, on average, the ablated model has on average around 80% of its patients becoming problem patients (the location of the white dot), and that 50% of the runs produced between approximately 75 and 85 %PP (the width of the black box). The grey violin shape is not like a bell curve, but instead has two bumps, characteristic of a bimodal distribution, giving us some extra information, namely many runs produce around 75 %PP, and many produce around 85 %PP, but comparatively few fall between those values. The first two violin plots show the overall distributions of %PP for each method over 100 runs, while the third shows the paired difference for each scenario in terms of %PS. The paired difference is approximately normally distributed with a mean %PS of 20.79% in favour of the new method and 95% confidence interval (19.55, 22.05).

The results comparing our model with an ablated version capable of direct preemption appear in figures 6.2 to 6.4. Confidence intervals were omitted for clarity, but were never larger than ±1%. We used a four factor ANOVA to compare the %PP of the two models, with the three simulation parameters and the preemption model as predictors. All predictors were highly significant ($p < 10^{-16}$), with a mean difference in favour of the new model of roughly 4%PS averaged across all parameter settings. Nearly all interaction terms were significant as well. In particular, we found that larger numbers of medical resources and patients tended to produce larger gains for the new system, up to some saturation point, while low initial patient health states produced gains as well.

Figure 6.5 shows the optimal conditions for our algorithm, in which there are many doctors (15) and there is a reasonable amount of time for patients to be treated (medium initial criticality), corresponding to the green line from figure 6.2. Error bars show 95% confidence intervals for the mean improvement. In this domain, especially under high congestion (400 patients) our system offers very significant advantages over the ablated model. The differences amounts to around 50 %PS when we have 400 patients for instance, which indicates that our method generates, on average, 200 fewer problem patients in a run than the ablated comparison system that is modeled on previous approaches[51].

6.6 Discussion

We have proposed a method for multi-agent resource allocation using TOC strategies to provide more detailed estimates of preemption costs, illustrated for the scenario of medical resource allocation. This serves both as a case study, demonstrating how to map a problem onto the models required by our system, and as a vindication of the system itself, which surpasses the ablated models used for comparison, and which are similar (though not identical) to the methods of earlier authors. **For practitioners** we have shown above that our system performed better than using worst-case bounds on preemption cost, when:

- There were many resources to choose from;
- There were many tasks for the system to process;

- Some efficient allocation which could serve most tasks prior to the deadline really did exist given the time allowed and the resources available.

Conversely, although the ablated system most similar to existing work did not perform well, a variant which supported direct preemption performed well when:

- There were very few resources;
- There were very few Tasks;
- Most tasks could not be serviced, even with an optimal schedule.

We suspect that these highly congested domains caused these early versions of agents in our system to over plan, which leads to incorrect estimates of preemption costs. The learning model discussed in chapter 5 is expected to mitigate this concern, but additional simulations are needed to confirm this. We conclude that the new system would be well suited for use in hospitals during MCI, and in other similar problem domains which require direct preemption and have both many users and many resources, possibly since these scenarios are more likely to contain cyclic dependencies.

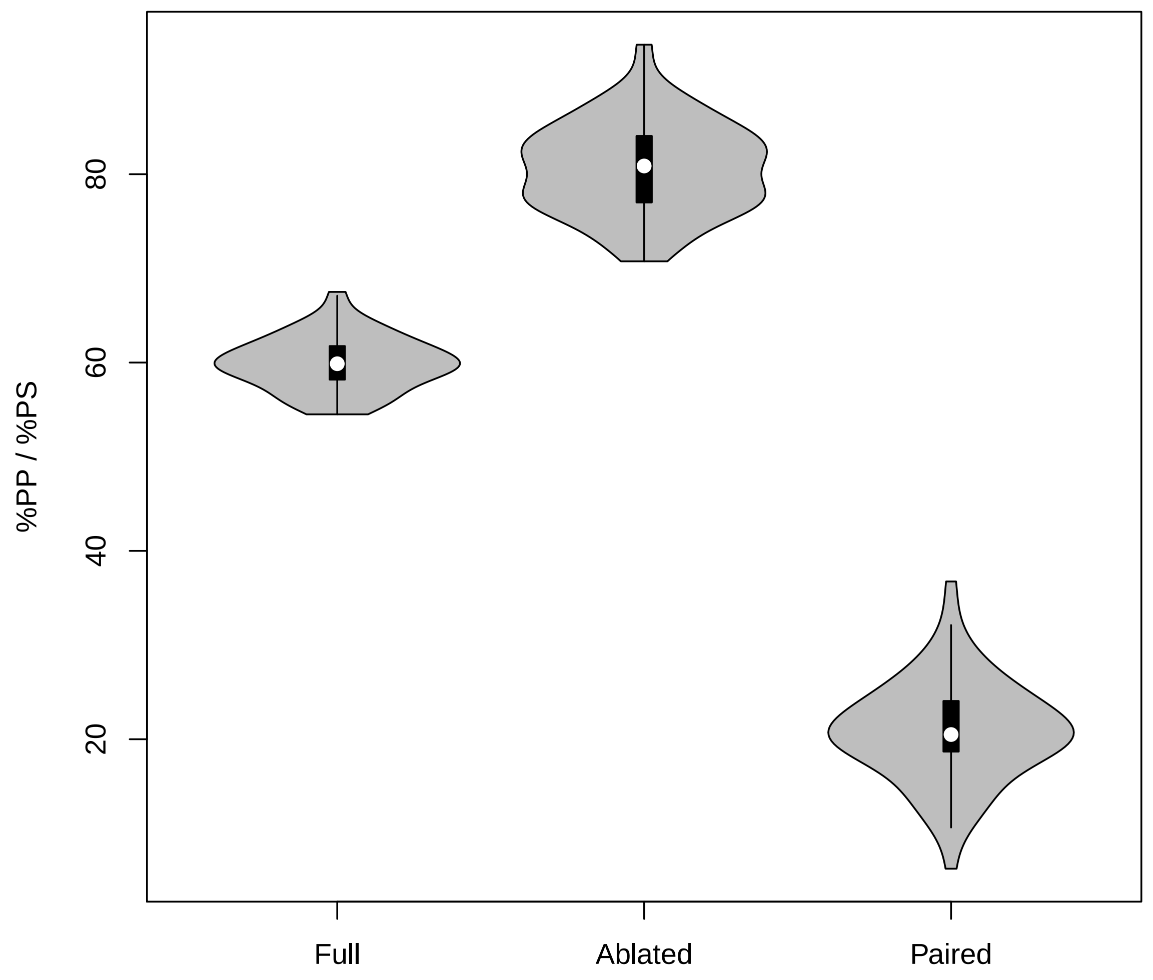


Figure 6.1: Violin plots [30] depicting the distribution of problem patients for the new system and an ablated version incapable of preemption. Also shown is the distribution of the pairwise improvement produced by using the new system on each scenario. Violin plots depict a box plot with a rotated kernel density plot (smoothed histogram) on either side.

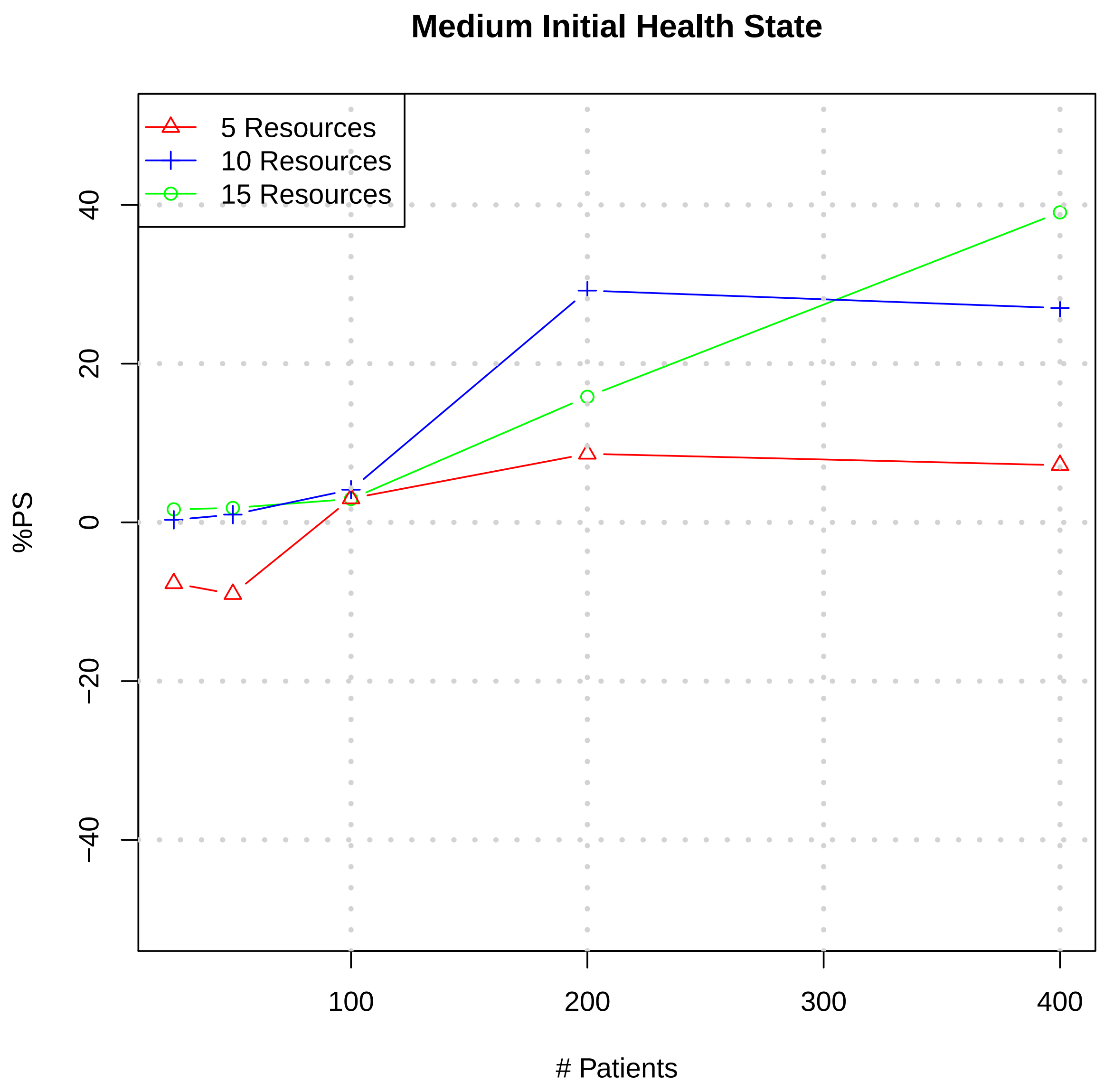


Figure 6.2: Mean improvements in %PS when initial health states are in $U(0, 50)$. Points above 0 show the advantage of using the new algorithm.

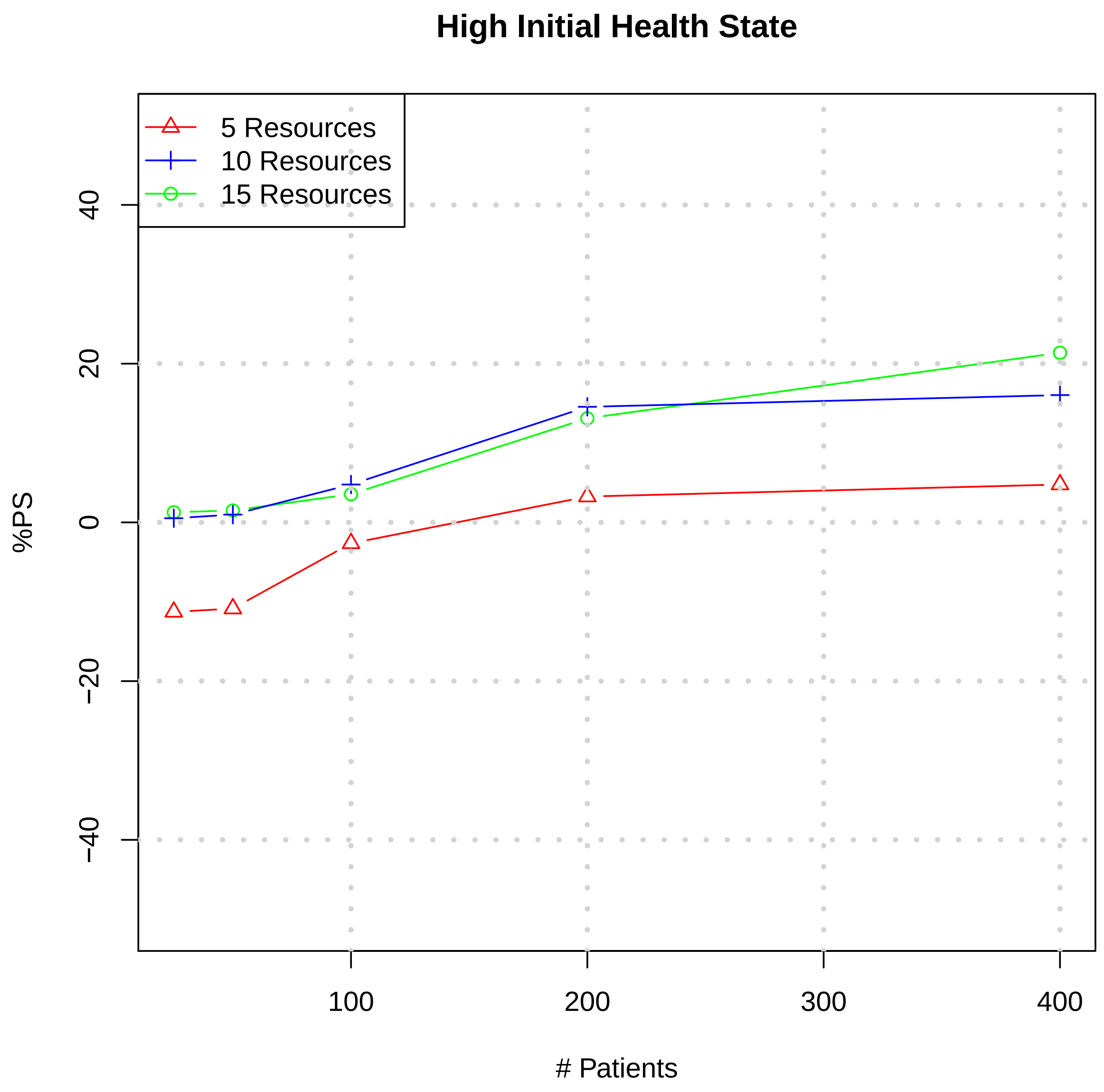


Figure 6.3: Mean improvements in %PS when initial health states are in $U(0, 100)$. Points above 0 show the advantage of using the new algorithm.

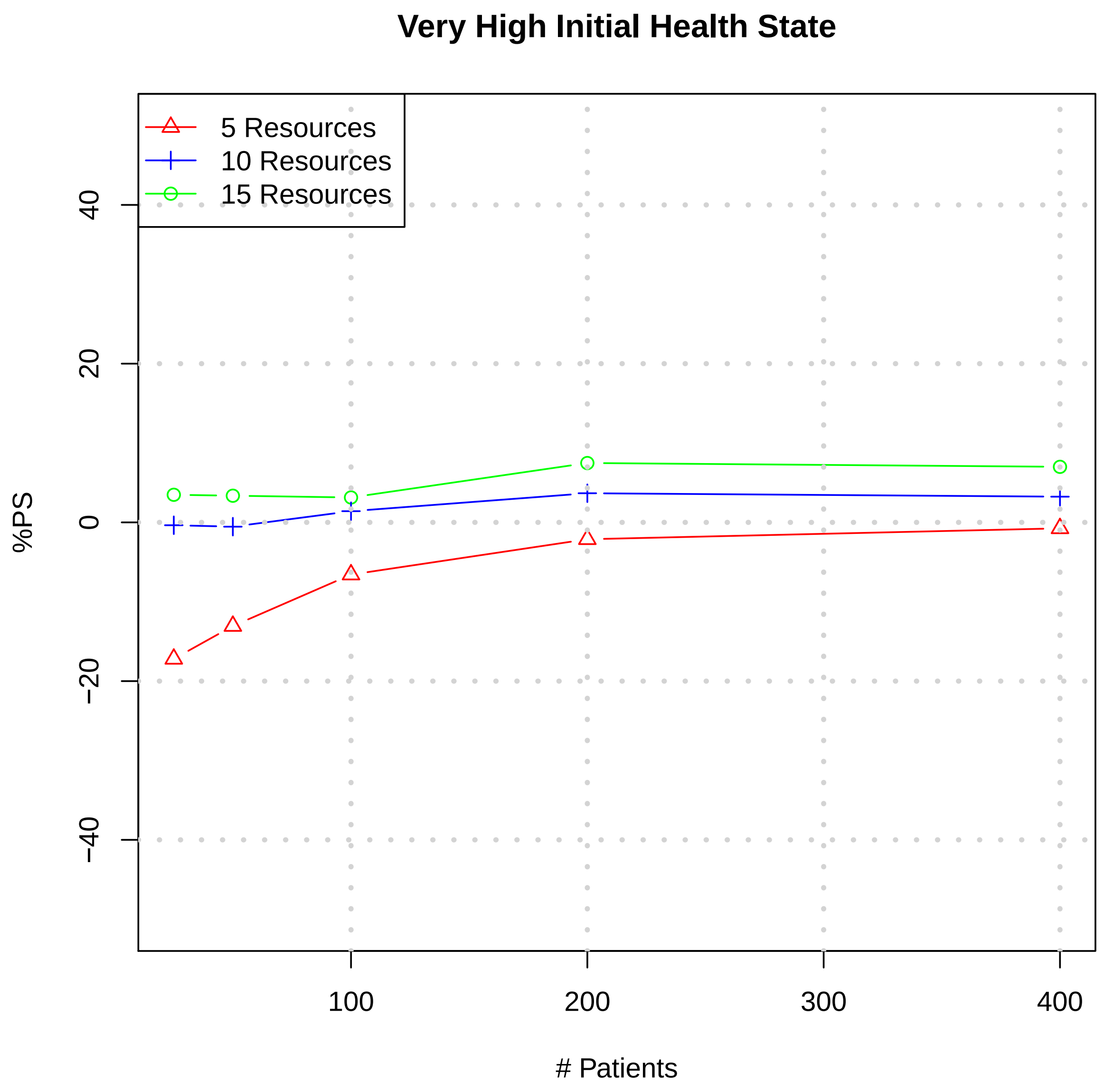


Figure 6.4: Mean improvements in %PS when initial health states are in $U(50, 100)$. Points above 0 show the advantage of using the new algorithm.

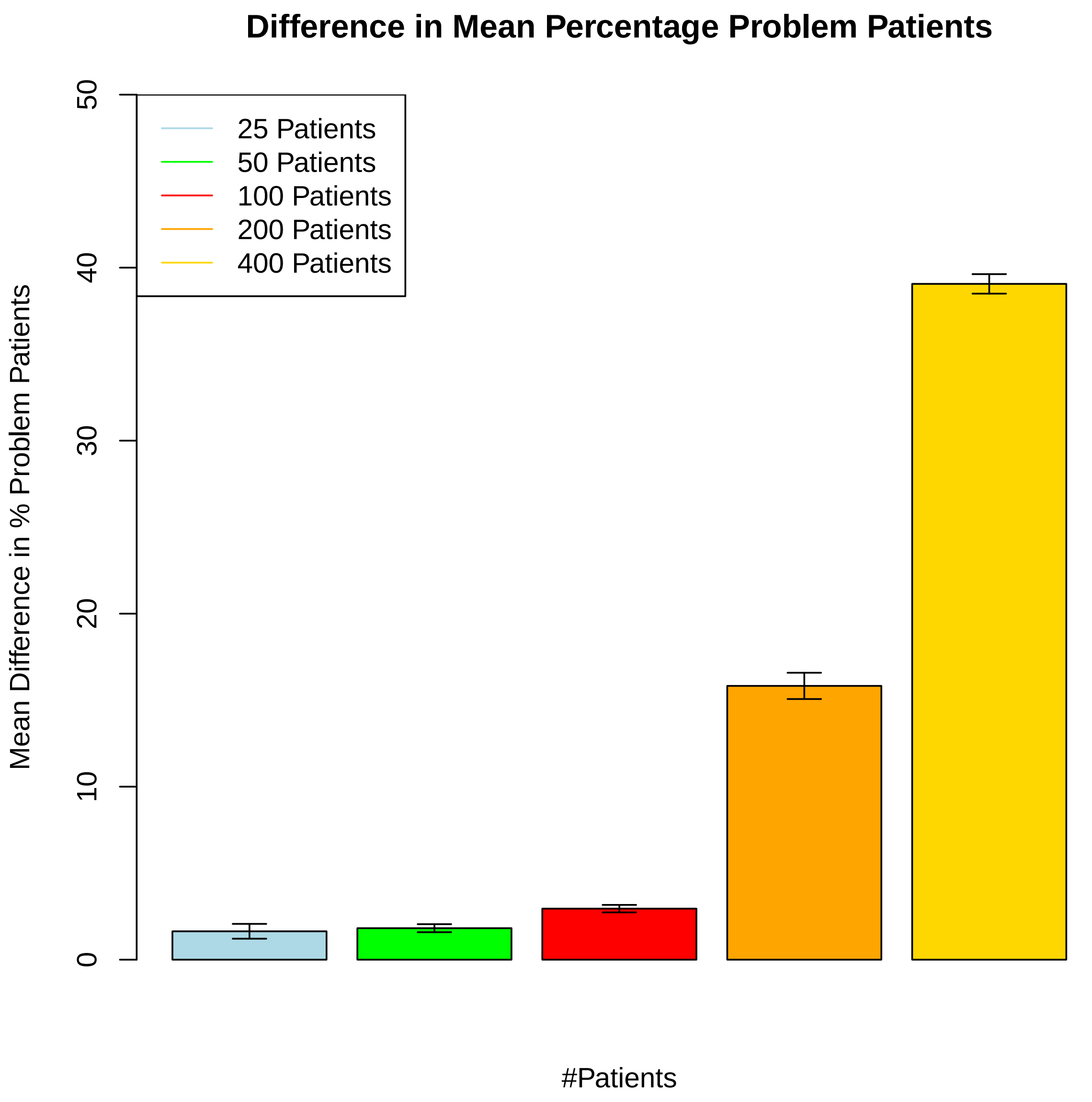


Figure 6.5: Mean improvements in %PS when initial health states are in $U(50, 100)$. Points above 0 show the advantage of using the new algorithm.

Chapter 7

Discussion

In this chapter, we discuss the benefits offered by the new system outlined in the preceding chapters, and contrast it with existing systems in similar domains. We begin the discussion by examining the new methods for TOC strategy generation which are presented in chapter 3. We then proceed to discussing the advantages of the system over existing approaches to medical resource allocation, again with an eye to mass casualty incidents, and finally conclude by comparing the system to other methods for solving generic resource allocation problems, and with a mention of the enhanced role played by proxy agents in our system.

7.1 TOC Strategy Generation

The research completed in Chapter 3 of the thesis constitutes a dramatic improvement to the selection of transfer of control strategies for adjustable autonomy multiagent systems: one that is able to produce optimal arrangements of TOC operations efficiently in highly dynamic environments and at the same time will still yield the determination of ideal times at which to transfer control to entities, as is the focus of the work of Cheng and Scerri et al.

In chapter 3, we presented two new approaches to the generation of transfer of control strategies. These new approaches build upon the work of three previous groups of authors, and we now discuss the advantages and drawbacks of using our new system as opposed to the existing ones.

We first discuss the approach to TOC strategy generation adopted by Jung [35]. Jung's methods were oriented toward medical resource allocation tasks, but did not consider the issue of resource scarcity in the same manner as we did here. Like in our system, Jung's system assumed fixed-width discrete time steps for TOC operations, meaning that each TOC operation took the same amount of time, and that the optimization problem for generating TOC strategies looked much the same as the one we assumed. Jung also placed restrictions on the types of partial transfers of control which were permissible within the system, in a fashion much like we did, and assumed that each doctor (resource) could be contacted at most once in the course of a single strategy. In fact, the constrained dynamic programming algorithm we presented in chapter 3 is capable (with very minor modifications) of solving exactly the form of TOC strategy generation that Jung was interested in.

All this said, we believe our approach to strategy generation should be strictly preferred to Jung's in the problem domain Jung's method was designed for. Jung utilized the enumerative approach we presented in section 3.3.1, which offers advantages over our constrained dynamic programming approach only in problems which have a tiny number of agents and resources, or in problems which have enormous amounts of processing power but almost no storage space. As neither of these scenarios is likely to be of interest, we advise future practitioners facing problems like those discussed in Jung's work to simply utilize our constrained dynamic programming algorithm instead of the enumerative approach.

Previous authors Cheng et al. [5] and Scerri et al. [63] examined problem domains which were less similar to ours, but also used more sophisticated approaches than Jung. In Scerri et al.'s E-ELVES project [63], agents built transfer of control strategies with real-valued, variable width, times for TOC operations. Thus, much of the effort involved in finding optimal TOC strategies was contained within the search for appropriate start and end times for each TOC operation in a strategy. Strategy optimization then took the form of a search, where possible TOC operations could be limited to a sort of Pareto front. For example, if it were known that from time 5 to time 10, Bob is busy and it is consistently better during that time to ask Margaret to make the decision instead, then we do not need to consider any strategy in which Bob is in control of decision making between time steps 5 and 10. Scerri et al. demonstrated empirically, in a real world deployment of a TOC strategy-based system [62] that the expected run times for TOC strategy generation were low, in part because optimal strategies were usually not very long - they contained a small number of TOC operations, each of which lasted for a large length of time, and, when valuation functions were continuous and differentiable with respect to time, finding the length of these operations is a constant time operation.

There are several reasons to suppose that this search-based approach would not necessarily be suitable in the resource allocation domain. First, the resource allocation domain is one which contains many deadlines and other forms of discontinuity in valuation functions.

These discontinuities will greatly increase the expense of finding optimal time lengths for TOC strategies [5], which then need to be decided though an empirical optimization process which is quite similar to the process used in our algorithms and those of Jung [35]. Second, we expect that in a resource allocation domain, especially in a medical resource allocation domain, strategies will tend to have more TOC operations of shorter lengths than those in the meeting scheduling domain examined by Scerri [63]. This is because if a resource is unlikely to respond to you quickly, it is likely that the resource is working on solving a task at present. In this case, you should prefer to ask someone else rather than waiting around (especially if your task is time dependent).

All that said, our system is capable of solving the sorts of TOC strategy generation problems used in the E-ELVES system [63], provided that the timestep size we use is sufficiently small, and of solving them with a comparable level of efficiency. Our unconstrained dynamic programming algorithm (3.2) is linear in the number of entities and the strategy length, which should be acceptable in all but the most demanding of real-time applications. Further, while the linear run time mentioned above is a worst-case bound in our algorithm, the search-based approach of the E-ELVES has a worst-case bound similar to the enumerative approach adopted by Jung [35]. Although scenarios in which the search algorithm actually needs large amounts of time are expected to be rare, in certain application domains, medical scheduling among them, we anticipate users preferring a slightly slower, but predictable, run time to one which is usually fast but might be exponentially slow.

Finally, we consider the work of Cheng [5] which added partial transfers of control to the E-ELVES model [63]. The addition of PTOCs provided considerably more flexibility to the system, as well as allowing for a more refined degree of user control. For example, Cheng [5] imagines an agent in a military aircraft which is capable of deciding whether or not to fire a missile against an impending threat. While the E-ELVES model would allow the agent only two options (fire the missile itself, or transfer decision making to the pilot), Cheng's model is capable of more nuanced actions, like asking the pilot for *permission* to fire. Like Jung's work on TOCs, Cheng's system utilized a detailed user model to weigh the costs of even these minimal interactions against the benefits. If the pilot is in the midst of a complex maneuver, and the system is reasonably certain that the target should be fired upon, then perhaps it would be preferable for the system to fire itself instead of asking.

Cheng utilized the search based methodologies of the E-ELVES system [63] when deciding both who to make a PTOC to and how long to wait for a reply. This proved to be an efficient method of strategy generation even in the presence of PTOCs [5], but again, was reduced to a form of 'empirical time step optimization' when valuation functions were not differentiable. PTOCs present a rather more complex problem in this case, since they have the potential to greatly increase the branching factor of the search. Our proposed method can mitigate this drawback for certain types of PTOCs in systems with non-differentiable valuation functions in a fashion similar to the speedups offered for the E-ELVES system above. Further, when actually generating the strategies, Cheng used a partially enumerative approach, starting with simpler strategies and moving to more complex ones, stopping if the gains from increasing complexity became small.

Additionally, heuristics which can stop the search prematurely, like those used by Cheng [5] can readily lead to suboptimal solutions. Previous authors [42] have shown that heuristic guided search will perform worse even than objective free searching on problems where its heuristic is *misleading*, and causes it to throughly search portions of the search space which do not contain good solutions to the optimization problem at hand. For example, we might imagine a problem in which there are 10 people we could transfer control over decision making to. Nine of them are inept yet eager, meaning they will produce a very poor quality decision (yet, still better than no decision at all), and are very likely to make the decision if we transfer control to them. The tenth person is highly competent, but is busy until 9 time steps in the future, and quite unlikely to answer. So skilled is this person that his or her decision is to be preferred over the inept persons' even if we wait nine time steps before it is made. The optimal strategy is therefore to wait for 9 time steps, and then transfer control to this agent. A deceptive heuristic for this problem would be using the expected utility of a presently generated strategy as a lower bound for the value of strategies extended from it. Since the inept people will produce a better decision than waiting, a search tree enumerating every possible permutation of these 9 FTOC operations will be created before we start to consider strategies which involve waiting 9 time steps before asking the competent person.

7.2 Medical Resource Allocation

The research completed in Chapter 6 of the thesis constitutes a substantial advancement for multiagent resource allocation, in domains such as mass casualty incidents: a system that is able to handle preemption of tasks. To date, efforts considering preemption were scarce and those available proposed unsophisticated, or overly conservative approaches. Our methods introduce a principled methodology that has been demonstrated to deliver a 40 percent improvement in the number of problem patients, when set against a very generous recasting of the most promising competitor, the model of Paulussen et al.

In this section, we discuss contributions made toward medical resource allocation using MARA techniques. We pick three authors whose systems are especially focused in this area, and contrast them with our system, arguing for the new system offering advantages over all of them in medical MARA domains.

In some sense, this work is the progeny of two recent systems for medical resource allocation tasks. Jung's [35] work on resource allocation for single agents provides insight into the use of TOC strategies and user modeling in the domain of emergency care, while Paulussen et al. [51] produced a medical resource allocation and scheduling system for multiple agents on which the negotiation protocol used in this work is based. That said, neither system offers a fully satisfactory solution to the medical resource allocation problem.

Jung's work [35] reasoned about which doctor an incoming emergency patient at a hospital should be assigned to. As discussed in the previous section, this was accomplished through the use of transfer of control strategies, which took into consideration the degree to which particular doctors were bothered and the abilities of doctors to actually treat the patient in question, which was dependant on the doctors' individual skill sets. This system is quite reasonable for use in a conventional medical setting, but the setting which served as a motivating factor in this work was that of a mass casualty incident. A distinctive difference between these two settings is the degree to which it is possible to share a medical resource. In Jung's framework, it is not unreasonable to 'time share' a resource by assigning many patients to a particular doctor and then allowing the doctor to decide who to treat and when, alternating among his/her assigned tasks as necessary. In a mass causality incident, we are more concerned with stabilizing patients who require immediate treatment, and there will tend to be many such patients for each doctor. Further, it may be necessary to interrupt a doctor who is in the midst of a consultation to treat an emergency patient. In these scenarios, we therefore expect that having doctors schedule patients will introduce avoidable inefficiencies into the allocations. Our system assigns doctors only one task at a time, and moves them to other tasks when required, eliminating this potential inefficiency. In addition, Jung's system does not account for the possibilities of processing several patients in parallel or for the possibility of having a patient switch doctors after the initial assignment, both of which are features of our system.

In contrast to Jung, Paulussen et al. [51] [50] consider the problem of patients attempting to acquire indivisible appointment times with a doctor. Each patient is assigned an agent which negotiates on their behalf by spending a sort of virtual currency in exchange for improved schedule spots. Like in our work, Paulussen et al. reason about the opportunity costs of taking a time slot from another patient in terms of each patient's health state and the expected improvements in health states arising from ownership of the timeslot, but, like in our work, this reasoning leads to the creation of preemption cycles. In Paulussen et al.'s system, only one agent is active at a time, and agents are fully myopic (that is, they are only thinking about what their immediately subsequent action would be, and picking that action using locally greedy reasoning). This factor prevents the agents from producing a reasonable resolution of the preemption cycles, necessitating the use of worst case estimates in their place. While maintaining many other aspects of Paulussen et al.'s system, we improve upon this element through the use of TOC strategies providing a substantial gain in performance.

A secondary issue is that the use of virtual currency in a medical resource allocation system has several undesirable properties which arise when new patients enter the problem midway through the allocation optimization. In particular, patients with relatively minor conditions who arrive early will tend to acquire excess currency from more urgent patients who arrive later, which can lead to them receiving higher quality care solely on the basis of fortunate early acquisition of resources. While this can be prevented by restarting the allocation process each time a new patient arrives, this property reduces the incrementalism of the system – partial solutions cannot be kept when the problem changes. As we would like this property in a fast paced environment like a mass casualty incident, this can be a significant drawback. Our system avoids this issue by eschewing the use of currency, allowing for incremental improvements.

Finally, we note that Paulussen et al. [50] do not consider the possibility of so-called 'direct' preemptions, wherein a patient preempts a human resource from another patient, rather than merely taking a timeslot. In such scenarios there are naturally extra costs, including both costs to the preempted patient (stopping in the middle of surgery is quite different from stopping in the middle of a checkup), and costs to the doctor (especially bother costs [10]). In contrast, our system is capable of modelling these costs through the inclusion of a detailed bother cost model for medical resources.

The final work considered here is that of Zhang et al. [83], who present a doctor-driven model, in some sense adopting the opposite approach of Paulussen et al. In Zhang's model, an agent whose patient is in need of treatment produces a certificate detailing their needs, and submits it to a sort of "patient market". The agents of doctors currently seeking a case compare their owners skills with those of the patient in question. The patient is then presented with a short list of these "bids", and may select a doctor of his or her choice. There are several factors which make this system unsuitable for MCI. First, Zhang's system fails to consider the possibility of preemption. Only doctors who are currently looking for patients are considered. Further, even if preemption were allowed and suitable information about the costs of preemptions gathered, Zhang's system places the final decision about which doctor to choose in the hands of self-interested patients. Possibly, these patients will be less inclined to worry about the well being of others when they are in life threatening situations themselves. For these reasons, we drew less inspiration from Zhang's approach than those of other authors, but note that this model offers a competing approach to a problem similar to the one adopted by our work, and could therefore form the basis of an interesting alternative avenue of research into the problem of preemption in medical resource allocation.

7.3 Generalized Resource Allocation

The research completed in Chapter 5 of the thesis constitutes a significant first step towards addressing generalized resource allocation: a system that demonstrates the value of learning techniques as part of the resource allocation process. In addition, the solution presented in this chapter achieves a significant goal left unattended in the work of Cheng: providing for a more effective role for proxy agents in multiagent systems. The coordination achieved between the plans of agents in our system is now dramatically enhanced with the role we assign to these proxy agents, as demonstrated in the validations offered within the chapter.

In this section, we examine the benefits of our new system over other generic systems for resource allocation.

Edwin & Cox studied generic resource allocation in the context of a multiagent system, and also examined the problem of resource preemption [20]. Their COMAS system proposes that an agent making a preemption request for a resource ask the agent currently in control for a measure of the value the agent places on holding the resource. If the preempting agent values the resource more than the reported value to the controlling agent, the preempting agent takes the resource. However, as noted by Cheng [5], COMAS utilizes only homogeneous resources, and so cannot be directly applied to doctor allocation during MCI, where there is a strong differentiation between the quality of different resources for different tasks. This also minimizes the need to consider the opportunity costs of preemption, since, in a scenario with homogeneous resources, the preempting agent's opportunity costs will be proportionate to those of the preempted agent. Our work is able to overcome this significant limitation of the COMAS system by providing a general purpose method for evaluating the opportunity costs of preemption in a system with heterogeneous resources.

Additionally, we note the advantages of our system over more conventional approaches to resource allocation. In particular, two broad categories of centralized approaches are of interest to us: optimization techniques and heuristic guided search. We take integer linear programming [41] as a canonical example of the former, while the latter tends to be highly problem specific [81]. As discussed above, these systems have seen widespread application in resource allocation problems, including scheduling [72] [4], budgeting [38], and location assignments [48] [54]. They are reasonably efficient in practice, and generate good solutions.

In spite of this, there are certain advantages the new system offers over these approaches. First, as a distributed system, our approach is free from the centralized points of failure which can reduce the reliability of centralized systems [40] [70] [71] [12]. This is important because, in a medical context for example, failure of a single system component can only impact a single patient, while in a centralized system, all patients could be impacted. Our system is also inherently incremental, allowing for ongoing gradual improvements in solution quality even when the problem is changing rapidly. In these scenarios, certain centralized approaches face a substantial problem, since their efficiencies and guarantees of optimality rely in some sense being sure that specific parts of the problem space are not worth examining further. If the problem changes, this assumption may no longer be true, requiring either restarting the optimization process, or accepting that the arrived upon solution may not match the present reality. In scenarios like the mass causality incidents mentioned earlier, this can be a substantial issue. We note however, that centralized systems which are content to find near optimal, rather than optimal, solutions are partially or entirely insulated from these concerns.

7.4 Coordinating TOC Strategies with the use of Proxy Agents

Finally, we discuss the enhanced role played by proxy agents in our system. In Cheng's model of Type IV agents [5], proxy agents acted as coordinating officials by tracking the bother produced by requests sent to a person from many different agents. Agents could optionally check their estimates of a person's bother against those of the proxy agent prior to submitted a TOC request in order to avoid wasting time with requests that were unlikely to be answered.

In our system the functionality and responsibilities of proxy agents are somewhat expanded. First, we make it mandatory for agents to check their models of a resource against those of the proxy before any request can be submitted. This is done because we anticipate that most agents will have at least partially inaccurate information most of the time, and so this small overhead can be used to avoid wasting an entire timestep on a TOC operation that is unlikely to succeed or which will produce a net decrease in overall utility. Second, proxy agents are tasked with providing complete estimates of the expected utility for proposed TOC operations by projecting the probable state of their resource into the future and reasoning about its ability to address particular tasks.

Proxy agents are in a unique position to accomplish these estimations. The estimations are dependent on information about the resource (which the proxy agent knows), information about the preempting agent's task (which said agent provides to the proxy), and information about the cost of the preemption to any task that the proxy's resource is currently working on (which the proxy determines by asking the task's associated agent for an estimate). Further, proxy agents can help agents learn the optimal plan length they should compute by providing feedback when the agent's plans have become stale. Consequently the expansion of the role of proxy agents is one of the vital factors in allowing our system to both coordinate TOC strategies and to facilitate learning, and represents an important contribution.

7.5 Summary

In summary, our new system offers a number of advantages over competing approaches.

With respect to TOC strategy generation we:

- Provide a strictly more efficient algorithm for solving the TOC strategy generation problems of Jung [35].
- Provide a new algorithm of greater or comparable efficiency for solving the TOC strategy generation problems in the E-ELVES system [63] when valuation functions are not differentiable.
- Provide a new algorithm of greater or comparable efficiency for solve TOC strategy generation problems when PTOCs [5] are limited to a specific form.

For medical resource allocation we:

- Combine the methods of Jung [35] and Paulussen et al. [51] to address concerns in both methods.
- Improve on the attempts by Paulussen et al. [50] to estimate the opportunity costs of a preemption using TOC strategies.
- Solve preemption in a patient-centric way, while noting that a doctor-centric methodology [83] might also be possible.

For resource allocation in general we:

- Offer improvements over the COMAS system [20] by allowing for preemptive MARA without requiring homogeneous resource types.
- Provide a fully decentralized system which is resilient to failure of single points unlike more conventional centralized systems like a linear programming solution [41].
- Provide a system capable of incremental and distributed improvement which is robust to changes in the problem, unlike existing centralized [41] and decentralized [51] systems.
- Expand upon Cheng's model of type IV proxy agents [5] to produce a more robust proxy agent with a clearer role in agent coordination.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis we have made a number of key contributions, which are summarized here.

The central contribution of this work is the design and construction of a new multiagent resource allocation system which reasons about preemption. Previous work on preemption in MARA produced myopic systems which could not reason as accurately about the opportunity costs of relinquishing a held resource. Our new approach is less myopic, and is capable of generating elaborate contingency plans for use following a preemption. Further, our new system utilizes simple learning algorithms to coordinate agents within the system. Using these algorithms we are able to dynamically adjust the foresight of agents in response to greater or lesser degrees of congestion and churn in the system. These properties allow agents to reason about the likelihood of preemptions and the likelihood of actually acquiring a new resource if theirs should be preempted. The system has the property of being 'Ex Ante Rational', meaning that rational agents would voluntarily use it before, but not necessarily after, gaining knowledge of their 'type'. This narrows the application of the system slightly, but, as we argue, still encompasses a large array of collective action problems, especially those in the health care domain.

We have also contributed a detailed evaluation of the proposed system through the use of simulated scheduling and resource allocation problems. This evaluation lent credence to our claims that the new system's hyperopic agents provided a measurable benefit. We found that, in general, agents which could build longer plans could outperform those who built shorter ones, but at the cost of some initial time spent assessing the problem domain before actions could be taken. In addition, we determined both that it was possible for plans to become too long, paralyzing agents with indecision; and that the optimal plan length fluctuated with the degree of congestion in the system. We used this information to provide several concrete recommendations for practitioners.

In addition, we offer several insights into the value of transfer-of-control strategies in the MARA domain. Since TOC strategies represent a plan of action for 'making a decision', and provide an easy framework from which to reason about the relative benefits of attempting to make the decision, there is a natural mapping to the MARA problem of making a plan of action for 'processing a task', with the estimate of the relative benefits being readily mapped to the opportunity costs of a preemption. Along with observing this relationship and utilizing it within our system, we perform a detailed analysis and comparison of several algorithms for the generation of TOC strategies. We present two new dynamic programming algorithms for use in TOC strategy generation, which offer significant gains in worst-case run time over existing approaches within certain problem domains. We provide both a detailed description of the assumptions required for the algorithm to work (including the central assumption that TOC operations are Markovian), and advice for practitioners, including a simple equation to facilitate practitioners in determining whether the second (constrained) dynamic programming algorithm is suitable for use in their problem domain. We also provide justification of the central Markovian assumption in the domain of resource allocation.

Finally, we compared an implementation of the new system to ablated variants which mimicked the behaviors of existing systems in the simulated medical problem domain of a mass casualty incident. We used this case study to demonstrate how real-world resource allocation problems could be modelled within the system, including the calibration of user models and the selection of appropriate parameters. Our validation confirms that the new system can offer substantial benefits over myopic approaches to MARA, in particular, by greatly reducing the number of simulated patients who were left unattended for too long (i.e. 'problem patients').

8.1.1 Summary

To summarize, we conclude with the following contributions:

- A methodology for multiagent resource allocation capable of supporting preemption of tasks with an approach that is:
  - Less myopic and thus better able to avoid inefficient allocations.

  - Contingent on reasoning about future opportunities if current resources are yielded at the present time.
  - More precise in its estimations due to the integration of learning both about likelihood of preemptions and likelihood of success in acquiring desired resources.
  - Is Ex Ante Rational, making it suitable for application in a wide array of resource allocation domains, especially medicine.

- A clarification of the valuable role that transfer of control strategies have in allowing agents to reason less myopically about relinquishing their current resources within multiagent resource allocation algorithms.

- A validation of the proposed methodology that is able to:
  - Calibrate the benefit or cost of trying to reason further into the future when generating transfer of control strategies.
  - Provide advice for practitioners about when the proposed transfer of control approach for reasoning about preemptions is beneficial, describing the properties of the application and its heuristics that need to apply.

- An approach for generating required transfer of control strategies more efficiently than has been achieved in the past under specified circumstances, with central insights:
  - That it is reasonable to assume a Markovian property for a certain class of adjustable autonomy scenarios.
  - That once a Markovian property is assumed it is possible to solve optimal subproblems within the TOC strategy generation process in order to eliminate repetitive work.

- A detailed case study of our proposed algorithms for multiagent resource allocation for the application area of allocating doctors to patients in a mass casualty incident scenario along with:
  - A clarification of desirable user model parameters and how to model them.
  - A validation of the algorithms as ones that deliver higher utility than competing approaches for resource allocation.

8.2 Future Work

Finally, we leave the reader with some interesting lines for future inquiry. These possibilities are sub-divided into three areas: improvements in efficiencies, improvements in the system, and more involved experimental designs.

8.2.1 Improvements in Efficiencies

Although the system we utilized in this work is, by and large, an a efficient one, there remain several areas for improvement.

First and foremost is the possibility for massive parallelization of the system. At present, agents operate in discrete timesteps, but in a random order within each step. To improve both the speed and realism of simulations, agents should instead be assigned to separate system threads which operate in parallel. On modern hardware, we anticipate an increase in processing speed by a factor of 8 should such changes be implemented. To introduce such parallelization, agents, tasks, and resources would need to undergo modifications in order to make them thread safe. Any function present in the three systems which is used for communication would need to be secured with semaphores. Additional care might be required to ensure that agents who are in the midst of generating a new contingency plan report reasonable values if asked for a preemption cost estimate during the process.

Alternatively, there exists a promising opportunity for parallelization in the most computationally expensive portion of the system: TOC strategy generation. Since a large number of agents may be simultaneously generating new strategies, and thus, performing identical sequences of operations on different data, a natural opportunity for vectorization of the code exists. Modern graphics processing units (GPUs) are capable of terraflop speed computation on such problems, offering a hundredfold speedup over conventional hardware. While converting the TOC strategy generation code to run on a GPU is a somewhat involved process, it is also one that could provide significant benefits without the added difficulties of inter-thread communications.

8.2.2 Improvements in the System

There are several avenues for improvement of the model, especially in the sophistication of the learning algorithms, the modelling of task constraints, and inter-agent communications.

As discussed in chapter 5, the learning algorithms utilized in this work are really just simple placeholders, and their performance can be improved. In particular, we found that on small problem instances, agents wasted a large amount of time learning reasonable parameter values before useful work could be done. One obvious avenue for future work is thus to improve the underlying learning algorithms and try to reduce this learning time as much as possible. This might be accomplished by providing more detailed feedback to the plan length and congestion models in the form of a variable cost per interaction. In the case of the plan length model, we might impose larger costs for interactions which occur earlier in an agent's plan, and smaller ones for costs which occur near the end of a plan's execution. For the congestion model, we might impose larger costs for interactions which happen close together or costs as a function of the net benefit gained by the preemption.

In addition, we could adopt a more sophisticated learning model. A first step might be a second order model which could adapt its rate of capacitive decay in response to the environment. Thus, for example, agents who tend to experience long gaps between interactions might have longer memories than those who experience very frequent interactions. We could also abandon the capacitive memory model entirely, and instead opt for a parameter estimation algorithm using some form of gradient descent. Finally, in the case that the user model is unknown, but the user can be observed, a model-free optimization algorithm like eurequa [64] could be used to construct a useful predictive model from raw data.

A secondary benefit to such an implementation would be the ability to more throughly determine the effects of learning on the system. In chapter 5 we found that our system was incapable of solving large scale resource allocation tasks without the aid of learning - learning is required to provide an accurate estimate of preemption costs. A side effect of this finding was that we were unable to compare our system with an ablated version to precisely quantize the improvements due to learning. A more sophisticated model might be more amicable to partial ablation, providing greater insight into this issue.

A second avenue for improvement is in the modelling of task constraints. Consider a task that needs two distinct operations to be accomplished before it can be processed, but the nature of the second task will not be known until the first task is complete. For example, a patient might need to have a diagnostic test performed. The test result will indicate a need for surgery in 25% of cases, medication in 50% of cases, and no further treatment in the remaining 25% of cases. A straightforward method of modelling this task is to first add the test to the system as a task, and then, once the test is complete, to add the second component as a new and separate task if necessary. It seems likely however, that a more sophisticated method could produce better results. For example, a shrewed agent might attempt to acquire appointments with doctors for all likely outcomes of the test while the test is being run. It could then cancel appointments that were unnecessary. Determining the preemption costs associated with such ‘possibly needed’ resources appears a non-trivial, yet interesting and potentially fruitful line of research.

Examining the possibilities for more efficient performance in scheduling MARA tasks is also quite an interesting possibility. At present the system both assumes a finite number of resources and does not assume that they are interchangeable. As a consequence of this, agents do not engage in resource ‘swaps’, wherein an agent whose resource was preempted receives the old resource of the preempting agent, since there is no reason to suppose that this second resource is likely to be useful to the preempted agent. Scheduling problems can however contain many similar resources - timeslots in a schedule. In these situations resource swaps might be a prudent approach. An alternate approach might be to allow the creation of new resources by inserting extra spots into the middle of the schedule and sliding the subsequent processing times back. The cost of such an operation might entail summing over a number of small opportunity costs rather than a single large one, but the exact implications of this are not entirely clear, and are worthy of examination.

The combination of several of the above features into a single system represents a substantial challenge in and of itself. For example, if more elaborate task constraints were introduced then the additional scheduling features can become extremely challenging to consider in an optimal fashion. An agent which holds spots in 20 different schedules at once arranged so that the topological ordering of its tasks is satisfied (i.e. each task only reaches the front of its schedule after all its prerequisite tasks will have finished processing) could have an extremely difficult time estimating the true cost of preempting or delaying one of these schedule spots, since this could entail re-ordering some or all of the other 20 tasks across many different schedules. A learning model for such multi-faceted problems also remains unclear – it would certainly need to be multidimensional in scope, but the resulting ‘curse of dimensionality’ [3] would likely render the initial learning period for such a model exponentially longer than the periods observed here. Nevertheless, the combination of several of the above features remains a compelling problem because it would greatly increase the scope of resource allocation problems to which the system could be applied.

Finally, it would be useful to explore non-utilitarian measures for optimization. For example, we might wish to consider the risk-reward tradeoffs of individual users of the system, or aggregate measures like the Nash Product from chapter 2 which consider the fairness of allocations to individual members. The exact implications of these alternative measures with respect to preemption are unclear, but interesting. Additionally, we might look at scenarios where agents do not have a common scale for expressing their utilities, violating the assumption we made in chapter 4. In this situation, the system needs to elicit some information from the user in order to facilitate translation between different utility scales. Alternatively, it might be interesting to attempt the formulation of a preemption cost estimation that is independent of the agent's individual utility scales.

8.2.3 Additional Experiments

Several additional experiments would be interesting to run.

In the work described here, we have always assumed that the system has access to perfect information about the world. In practice this is unlikely, and so we would like to produce realistic sources of noise. In the mass causality scenario described in chapter 6 for example, we can imagine several realistic sources of noise:

- Patients' health states should have Gaussian noise, and occasional interruptions of service, representing both a noisy and unreliable signal.
- Patients should occasionally have a condition which differs from the one reported to the system. For example, a critical patient might appear to be a routine one as a result of misdiagnosis.
- Doctors' skill levels should occasionally differ from those reported to the system. For example, a particular doctor might receive poor performance evaluations from a manager despite great skill, or might have undocumented skills picked up in previous occupations.
- Doctors' bother levels (and bother cost increase functions) should differ from those reported to the system.
- Agents might be required to utilize noisy communication channels, which drop requests or which direct them to incorrect resources occasionally, simulating a poorly performing
- Agents might fail unexpectedly, requiring the system to adapt and recover. network.

Another interesting experiment would be to quantify the closeness of our system's solutions to the optimal on a more diverse set of problem, and in particular, to determine the distributions of outcomes for tasks (i.e. not just the mean disutility per task, but the variance as well). Comparison of the system to a centralized approach at each time step could provide compelling evidence of convergence.

Finally, in this work we have assumed that the utility functions for individual transfer of control operations are actually discrete, matching the assumptions of our model. It would be interesting to allow functions to be continuous, and then to find optimal starting and stopping times in a number of different ways. First, we could vary the granularity of a discrete optimization process, by gradually shrinking Δt, and determining the values at which errors in the estimated utility become sufficiently small. Second, we could use stochastic dynamic programming [2] to repeatedly generate strategies with different stopping points in order to find nearly optimal strategies quickly. Finally, we could modify the new TOC strategy generation techniques shown here to incorporate the techniques of Cheng [5] for finding optimal starting and stopping point for continuous time TOC operations. It is not immediately clear which of these methods would be preferable, both in terms of run times and approximation accuracies.

Appendices

Appendix A

User Modelling for a Medical Application

In this work, we computed UUF based on the evaluation of the total expected change in the health state of the patient if the doctor agreed to treat them. That is, a high possibility of a doctor making a serious difference in a patient's health was used as a proxy for the doctor's willingness to actually treat the patient. This could have several possible meanings. For example, a doctor who lacked the skill to treat a particular patient effectively would have a higher UUF than one who had the skill. Similarly, the doctor would be more willing to help a dying patient who he or she could save than a mostly well patient or one past the point of help, both of which seem reasonable as modelling decisions. Formally, when a doctor received a request for treatment, we computed UUF based on four factors of import.

Suppose we have a doctor d and patient p. P asks d for treatment, which would start at t_1 and continue until the patient recovers. The time at which the recovery is projected to take place if the patient is left untreated is called t_2. We denote the present criticality of a patient p at time t, which changes as they are treated or deteriorate, as $crit(p,t)$. The first factor is the improvement in p's condition which would be experienced as a result of d's treatment:

$$help(d,p,t_1,t_2) = \int_{t_1}^{t_2} \Big[\frac{dCrit}{dt}(has(p), crit(p,t)) - treat(d, has(p))\Big]dt \tag{A.1}$$

The second factor is the estimated value of any treatment p may be receiving already from a doctor d_2 who would be replaced by d. If d_2 does not exist, then the value of

$treat(d_2, has(p))$ is assumed to be 0. Otherwise, it is given by:

$$help(d_2, p, t_1, t_2) = \int_{t_1}^{t_2} \Big[\frac{dCrit}{dt}(has(p), crit(p,t))) - treat(d_2, has(p))\Big] dt \qquad \text{(A.2)}$$

The third factor is the expected quality of any new treatment acquired by a patient p_2 who is currently undergoing treatment by d, in the event that d disappears. This is calculated in a special fashion using the valuations of TOC strategies. Here we will refer to it as

$$ContigencyValue$$

and assume it to be zero if p_2 does not exist.

Finally, there is the expected loss of treatment for p_2, which is again, set to zero if p_2 does not exist.

$$help(d, p_2, t_1, t_2) = \int_{t_1}^{t_2} \Big[\frac{dCrit}{dt}(has(p_2), crit(p_2,t))) - treat(d, has(p_2))\Big] dt \qquad \text{(A.3)}$$

These terms can be combined to produce *criticality functions*. The first of these functions describes the sum of the expected criticality of patients p and p_2 at time t under the assumption that the request is denied. It is shown in equation A.4.

$$crit_{denied}(t) = [help(d, p_2, t_1, t) + crit(p_2, t_1)] + [help(d_2, p, t_1, t) + crit(p, t_1)] \qquad \text{(A.4)}$$

Likewise, the second describes the expected criticality of patient p at time t under the assumption that the request is approved, shown in equation A.5.

$$crit_{approved}(t) = [help(d, p, t_1, t_2) + crit(p, t_1)] \qquad \text{(A.5)}$$

The equation which describes the numerical UUF is:

$$UUF = \int_{t_1}^{t_2} \Big[crit_{denied}(t) - crit_{approved}(t)\Big] dt - ContigencyValue \qquad \text{(A.6)}$$

Which is conceptually, the area between the criticality-time curve manifested when the request is not accepted and the same curve when the request is accepted (assuming that $ContigencyValue$ represents the expected area under the criticality curve for $p2$ in the

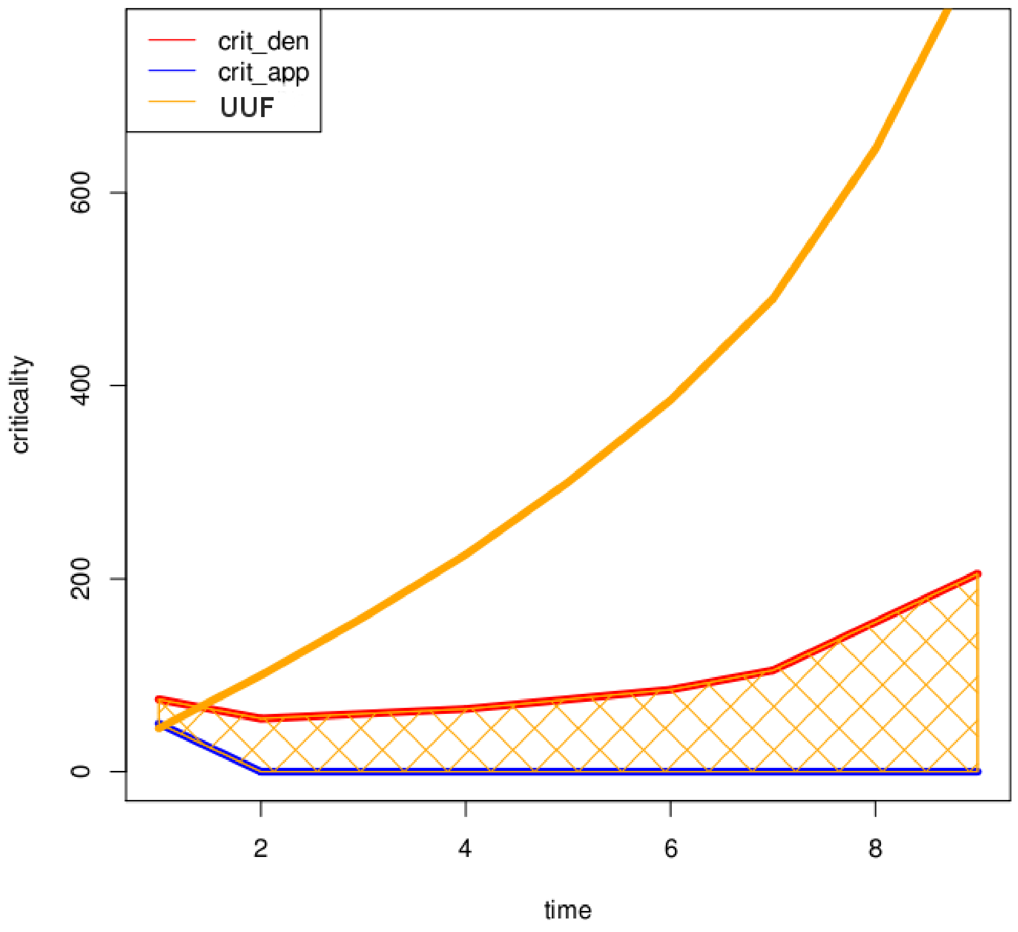


Figure A.1: A visual representation of numerical UUF, the orange curve corresponding to the area denoted with orange crosshatch. The blue and red curves represent the criticality of a patient under different assumptions, integrated over time.

event of a preemption - more on this later). A visualization of the UUF values is shown in figure A.1. The curves for $crit_{denied}$ and $crit_{approved}$ are shown. The area between them is highlighted by the cross hatch. The UUF curve shows the total area between the two criticality curves up to a certain point in time. We take the rightmost point on this curve as the final value.

For simplicity, and consistency with the terminology of Cheng [5] and Jung [35], we adopt a step function wrapper for the numerical UUF scores. A UUF of more than 100 is denoted as *low* UUF. A UUF between -100 and 100 is denoted as *medium*. A UUF of less than -100 is a *high*. Notice that a negative value corresponds to a 'high' unwillingness because it indicates that a doctor would actually cause harm by accepting the request, perhaps as a consequences of abandoning a critical patient to help one who is less badly off.

ASF is modeled as a function of how vital it is that the doctor remain uninterrupted when treating their current patient. A patient whose condition tests the limits of a doctor's skills is thus more distracting than a routine case. ASF is modified by the level of bother already experienced by the doctor. A doctor who is bombarded with requests is quite likely to disengage from the system, and so is modeled as having a lower ASF. This is measured as the ratio:

$$ASF = \frac{treat(d, has(p))}{(1 + \gamma(BSF)) \times (1 + \frac{dCrit}{dt}(has(p), initCrit(p)))} \tag{A.7}$$

if the doctor has a patient, and:

$$ASF = \frac{1}{|C|} \sum_{c \in C} \frac{treat(d, c)}{1 + \gamma(BSF)} \tag{A.8}$$

otherwise, where C is the set of all possible conditions patients could have.

A value of less than 1 is considered to be a *low* ASF. Values between 1 and 2 are *medium* ASF, while values greater than 2 constitute a *high* ASF. Thus, a doctor who is trying and failing to stabilize a patient is modeled as being quite unlikely to respond. A doctor treating a difficult patient, but meeting with success, is a bit more likely, while a doctor treating a patient who is well within their skill level is most likely to respond. γ is the *bother discounting factor*, a scaling parameter which we set to 0.01 for consistency with the choice of β used in this chapter's experiments. The reason we average in equation A.8 over all conditions is because we assume that a more experienced doctor (i.e. one with higher average skill) without a patient will be, on average, more attentive to requests in

UUF/ASF	High	Medium	Low
Low	Willing	Medium Willing	Neutral
Medium	Medium Willing	Neutral	Medium Unwilling
High	Neutral	Medium Unwilling	Unwilling

Table A.1: The stance adopted by doctors in response to a request for treatment as a function of their UUF and ASF ratings.

general, even when those requests are not specifically geared to their skillset. Similarly, all else being equal, a doctor with a greater degree of bother is less likely to be attentive when without a patient.

Following earlier work [35], UUF and ASF are added together to produce 5 distinct categories, summarized in table A.1.

These categories are utilized in the bother increase function. A request for which the doctor is unwilling produces an increase in bother of $BSF^{1.25}$. A request for which the doctor is willing produces an increase of $BSF^{0.75}$. All other categories produce an increase of BSF.

References

[1] M. Bauer, D. Dengler, and G. Paul. Instructible agents for web mining. In *Proceedings of Human-Computer Interaction -INTERACT'01*, pages 593–601, 2001.

[2] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming.* Springer, 2011.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning.* 2006.

[4] J. T. Blake, F. Dexter, and J. Donald. Operating room managers use of integer programming for assigning block time to surgical groups: A case study. *Anesthesia and Analgesia*, 94(1):143–148, 2002.

[5] M. Cheng. A hybrid transfer of control approach to designing adjustable autonomy multi-agent systems. Master's thesis, University of Waterloo, 2005.

[6] M. Cheng and R. Cohen. Reasoning about interaction in a multi-user system. In Liliana Ardissono, Paul Brna, and Antonija Mitrovic, editors, *User Modeling 2005*, volume 3538 of *Lecture Notes in Computer Science*, pages 189–198, 2005.

[7] M. Y. K. Cheng and R. Cohen. A hybrid transfer of control model for adjustable autonomy multiagent systems. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, pages 1149–1150, New York, NY, USA, 2005. ACM.

[8] Y. Chevaleyre, P.E. Dunne, U. Endriss, J. Lang, M. Lematre, J. Padget, S. Phelps, J. A. Rodrigues-Aguilar, and P. Sousa. Issues in multiagent resource allocation. *Informatica*, 30:3–31, 2006.

[9] Y. Chevaleyre, U. Endriss, S. Estivie, and N Maudet. Multiagent resource allocation with k-additive utility functions. In *In Proc. DIMACS-LAMSADE Workshop on Computer Science and Decision Theory, Annales du LAMSADE 3*, pages 83–100, 2004.

[10] R. Cohen, M. Y. K. Cheng, and M. W. Fleming. Why bother about bother: Is it worth it to ask the user? In *AAAI05 Fall Symposium on Mixed-Initiative Problem-Solving Assistants*, 2005.

[11] R. Cohen, H. Jung, M. Fleming, and M. Y. K. Cheng. A user modeling approach for reasoning about interaction sensitive to bother and its application to hospital decision scenarios. *User Modeling and User-Adapted Interaction*, 21:441–484, 2010.

[12] B. F. Cooper and H. Garcia-Molina. Peer-to-peer resource trading in a reliable distributed system. In *Peer-to-Peer Systems*, pages 319–327, 2002.

[13] G. B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, 1963.

[14] K. Decker and J. Li. Coordinating mutually exclusive resources using gpgp. *Autonomous Agents and Multi-Agent Systems*, 3:133–157, 2000. 10.1023/A:1010074611407.

[15] K. Decker and Jinjiang Li. Coordinated hospital patient scheduling. In *Multi Agent Systems, 1998. Proceedings. International Conference on*, pages 104 –111, jul 1998.

[16] M. E. desJardins, E. H. Durfee, C. L. Jr. Ortiz, and M. J. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 20:13–22, 1999.

[17] J. A. Doucette and M. I. Heywood. Novelty-based fitness: An evaluation under the Santa Fe trail. In *Proceedings of the 13th European Conference on Genetic Programming*, pages 50–61, 2010.

[18] K. Dresner and P. Stone. Multiagent traffic management: A reservation-based intersection control mechanism. In *The Third International Joint Conference On Autonomous Agents and Multiagent Systems (AAMAS 04)*, 2004.

[19] E.H. Durfee and V.R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. 21(5):11671183, 1991.

[20] G. Edwin and M.T. Cox. Resource coordination in single agent and multiagent systems. In *Proceedings of the 13th IEEE conference on Tools with Artificial Intelligence*, pages 18–24, 2001.

[21] S. Einav, L. Aharonson-Daniel, C. Weissman, H. R. Freund, and K. Peleg. In-hospital resource utilization during multiple casualty incidents. *Annals of Surgery*, 243(4):533–540, 2006.

[22] S. Einav, Z. Feigenberg, C. Weissman, D. Zaichik, G. Caspi, D. Kotler, and H. R. Freund. Evacuation priorities in mass casualty terror-related events: implications for contingency planning. *Annals of Surgery*, 239(3):304–310, 2004.

[23] G. Fischer. User modeling in human-computer interaction. *User Modeling and User-Adapted Interaction*, 11:65–86, 2001.

[24] R. A. Fisher. The correlation between relatives on the supposition of mendelian inheritance. *Philosophical Transactions of the Royal Society of Edinburgh*, 52:399–433, 1918.

[25] M. Fleming. *Reasoning about Interaction in Mixed-Initiative Articial Intelligence Systems*. PhD thesis, University of Waterloo, 2003.

[26] M. Fleming and R. Cohen. A decision procedure for autonomous agents to reason about interaction with humans. In *Proceedings of the AAAI 2004 Spring Symposium on Interaction between Humans and Autonomous Systems over Extended Operation*, pages 81–86, 2004.

[27] L. Friha, P. Queloz, and C. Pellegrini. A decomposition method for hospital scheduling problems. In *Workshop on Industrial Constraint-Directed Scheduling*, 1997.

[28] M. Hannebauer and S. Müller. Distributed constraint optimization for medical appointment scheduling. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, pages 139–140, New York, NY, USA, 2001. ACM.

[29] H. Hexmoor and C. Castelfranchi. *Agent Autonomy*. Kluwer Academic Publishers Group, 2003.

[30] J. L. Hintze and R. D. Nelson. Violin plots: a box plot-density trace synergism. *The American Statistician*, 52:181–184, 1998.

[31] A. Hirshberg. Multiple casualty incidents lessons from the front line. *Annals of Surgery*, 239(3):322–324, 2004.

[32] E. Horvitz, A. Jacobs, and D. Hovel. Attention-sensitive alerting. In *Proceedings of Uncertainty in Artificial Intelligence (UAI'99)*, 1999.

[33] E. J. Horvitz, J. S. Breese, and M. Henrion. Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning*, 2(3):247 – 302, 1988.

[34] B. Jaumard, F. Semet, and T. Vovor. A generalized linear programming model for nurse scheduling. *European Journal of Operational Research*, 107:1–18, 1998.

[35] H. Jung. Reasoning about benefits and costs of interaction with users in real-time decision making environments with application to healthcare scenarios. Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, 2010.

[36] C. A. Kahn, C. H. Schultz, K. T. Miller, and C. L. Anderson. Does start triage work? an outcomes assessment after a disaster. *Annals of Emergency Medicine*, 54(3):424–430, 2009.

[37] R. M. Karp. Reducibility among combinatorial problems. In *M. Jnger et al. (eds.), 50 Years of Integer Programming 1958-2008*, 2010.

[38] A. J. Keown and J. D. Martin. An integer goal programming model for capital budgeting in hospitals. *Financial Management*, 5(3):28–35, 1976.

[39] S. Kim and P. K. Varshney. Adaptive load balancing with preemption for multimedia cellular networks. In *Wireless Communications and Networking*, volume 3, pages 1680 – 1684, 2003.

[40] K. Lai1, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman. Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent and Grid Systems*, 1:169–182, 2005.

[41] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[42] J. Lehman and K. O. Stanley. Novelty search and the problem with objectives. In *Genetic Programming Theory and Practice IX*, 2011. in press.

[43] Wolfram Alpha LLC. *Wolfram Alpha*. 2010.

[44] R. T. Maheswaran, M. Tambe, P. Varakantham, and K. Myers. Adjustable autonomy challenges in personal assistant agents: A position paper. In *Agents and Computational Autonomy, M. Nickles et al, eds*, 2004.

[45] M. F. McTear. User modelling for adaptive computer systems: a survey of recent developments. *Artificial Intelligence Review*, 7:157–184, 1993. 10.1007/BF00849553.

[46] J. S. Mill. *Utilitarianism (7th ed.)*. Longmans, Green, and Co., 1879. Retrived 2012 from Project Gutenberg: http://www.gutenberg.org/files/11224/11224-h/11224-h.htm.

[47] J. J. et al. Mistovich. *Brady Prehospital Emergency Care Sixth Edition*. Prentice Hall Health, 2000.

[48] R. L. Morrill and P. Kelley. Optimum allocation of services: An hospital example. *The Annals of Regional Science*, 3:55–66, 1969.

[49] K. Myers and D. Morley. Policy-based agent directability. In *Agent Autonomy, H. Hexmoor and C. Castelfranchi eds.*, 2003.

[50] T. O. Paulussen, N. R. Jennings, K. S. Decker, and A. Heinzl. Distributed patient scheduling in hospitals. In *International Joint Conference on Artificial Intelligence*, volume 18, pages 1224–1232, 2003.

[51] T. O. Paulussen, A. Zller, A. Heinzl, A. Pokahr, L. Braubach, and W. Lamersdorf. Dynamic patient scheduling in hospitals. *Agent Technology in Business Applications*, 2004.

[52] J. Pita, M. Jain, F. Ordnez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus. Armor security for Los Angeles international airport. In *AAAI Intelligent Systems Demonstrations*, 2008.

[53] D. Poole and A. Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, 2010.

[54] S. Rahman and D. K. Smith. Use of location-allocation models in health service development planning in developing nations. *European Journal of Operational Research*, 123:55–66, 2000.

[55] S. U. Randhawa and D. Sitompul. A heuristic-based computerized nurse scheduling system. *Computers & Operations Research*, 20(8):837–844, 1993.

[56] J. Rawls. *A Theory of Justice*. Belknap, 1971.

[57] A. Rogers, E. David, J. Schiff, and N.R. Jennings. The effects of proxy bidding and minimum bid increments within ebay auctions. *ACM Transactions on the Web*, page Article 9, 2007.

[58] G. Rogow. Rise of the (market) machines. 2009.

[59] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2003.

[60] T.W. Sandhlom and V. R. T. Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94:99–137, 1997.

[61] T. Sandholm. Contract types for satisfying task allocation. In *AAAI Spring Symposium Series, Satisfying Models*, 1998.

[62] P. Scerri, D. V. Pynadath, and M. Tambe. Towards adjustable autonomy for the real world. *Journal of Artificial Intelligence Research*, 17:171–229, 2002.

[63] P. Scerri, D. V. Pynadath, and M. Tambe. Why the elf acted autonomously: Towards a theory of adjustable autonomy. In *Proceedings of AAMAS'02*, 2002.

[64] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81 – 85, 2009.

[65] D. Schreckenghost, R. P. Bonasso, D. Kortenkamp, S. Bell, T. Milam, and C. Thronesbery. Adjustable autonomy with nasa procedures. In *Proceedings of 9th International Symposium on AI, Robotics and Space*, 2008.

[66] N. Schurr, J. Marecki, N. Kasinadhuni, M. Tambe, J. P. Lewis, and P. Scerri. The defacto system for human omnipresence to coordinate agent teams: The future of disaster response. In *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2005.

[67] N. Schurr, J. Marecki, and M. Tambe. Improving adjustable autonomy strategies for time-critical domains. In *Proceedings of the 8th Int. Conf. on and Castelfranchi*, pages 353–360, 2009.

[68] N. Schurr and M. Tambe. Using multi-agent teams to improve the training of incident commanders. In Michal Pchouek, Simon G. Thompson, Holger Voos, Marius Walliser, Stefan Brantschen, Monique Calisti, and Thomas Hempfling, editors, *Defence Industry Applications of Autonomous Agents and Multi-Agent Systems*, Whitestein Series in Software Agent Technologies and Autonomic Computing, pages 151–166. Birkhuser Basel, 2008.

[69] B. Settles. Active learning literature survey. *Computer Sciences Technical Report 1648, University of WisconsinMadison*, 2010.

[70] W. Shen, L. Wang, and Q. Hao. Agent-based distributed manufacturing process planning and scheduling: A state-of-the-art survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 26:563–577, 2006.

[71] A. Silberschatz and P. Galvin. *Operating system concepts*. Wiley India Pvt. Limited, 1994.

[72] D. Sitompul and S. U. Randhawa. Nurse scheduling models: a state-of-the-art review. *Journal of the Society for Health Systems*, 2(1):62–72, 1990.

[73] C. Starmer. Developments in non-expected utility theory: The hunt for a descriptive theory of choice under risk. *Journal of Economic Literature*, 38:332–382, 2000.

[74] R. Tamblyn, A. Huang, L. Taylor, Y. Kawasumi, G. Bartlett, R. Grad, A. Jacques, M. Dawes, M. Abrahamowicz, R. Perreault, N. Winslade, L. Poissant, and A. Pinsonneault. A randomized trial of the effectiveness of on-demand versus computer-triggered drug decision support in primary care. *Journal of the American Medical Informatics Association*, 15:430–438, 2008.

[75] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009.

[76] J.D. Ullman. Np-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393, 1975.

[77] H. van der Sijs, J. Aarts, A. Vulto, and M. Berg. Overriding of drug safety alerts in computerized physician order entry. *Journal of the American Medical Informatics Association*, 13:138–147, 2006.

[78] I. Vermeulen, S. Bohte, K. Somefun, and H. Poutr. Multi-agent pareto appointment exchanging in hospital patient scheduling. *Service Oriented Computing and Applications*, 1:185–196, 2007. 10.1007/s11761-007-0012-1.

[79] M. P. Wellman and P. R. Wurman. A trading agent competition for the research community. In *IJCAI-99 Workshop on Agent-Mediated Electronic Commerce*, 1999.

[80] S. J. Weng, G. M. T. Wu, and J. Fowler. Distributed resource allocation for healthcare systems. In *Proceedings of the IEEE International Conference on Service Operations and Logistics, and Informatics*, pages 1078 – 1083, 2008.

[81] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[82] T. Wu, N. Ye, and D. Zhang. Comparison of distributed methods for resource allocation. *International Journal of Production Research*, 43(3):515–536, 2005.

[83] X. Zhang, H. Xu, and B. Shrestha. Developing multi-agent systems with automatic agent generation and dynamic task allocation mechanisms. In *Proceedings of AAMAS07*, pages 1254–1256, 2007.