Scheduling for Humans and Team Size Issues in Multirobot Supervisory Control

Sandra Mau

CMU-RI-TR-07-20

Submitted in partial fulfillment of the
requirements for the degree of
Master of Science in Robotics.

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

June 14, 2007

Thesis Committee:
John Dolan, Advisor
Illah Nourbaksh
Kristen Stubbs

Contents

Abstract

This body of research describes efficient utilization of human time by two means: prioritization of human tasks and maximizing multirobot team size. An efficient scheduling algorithm for multirobot supervisory control (dSSPT) is proposed which schedules tasks such that a mission is completed faster. This algorithm is superior to existing algorithms by prioritizing human tasks such that robots can regain autonomous control sooner. In simulations of a single resource (e.g. human) scheduling problem, it is found that downtime is lower for dSSPT and the rate of human task completion is faster compared to other standard or similar algorithms. In simulations of a multirobot area surveying problem, we show that the rate of area coverage is much higher using dSSPT compared to first-in-first-out(FIFO).

This work also looks at maximum multirobot team size with the notion that handling more robots on the team can potentially mean more work gets done by the robots without increasing human time. The factors that affect team size are examined mathematically and verified through simulation. It is found that variance in interaction time between humans and robots, variance in neglect time during which a robot is autonomous, and the use of different scheduling algorithms can all affect the maximum number of robots a human can manage on a team. Another significant finding related to maximum team size is that the size is always the same or higher than an often-cited estimate known as fan-out. Since fan-out is derived from an ideal, average case, it turns out that the upper bound on team size predicted by the fan-out equation is actually a lower bound on the maximum team size for any practical situation (i.e, where task lengths and periodicity may vary or when robots are heterogeneous).

Chapter 1

Introduction

It is a common adage that time is money and in multirobot applications, this becomes all the more apparent. For example in multirobot planetary exploration, the Spirit and Opportunity Martian rover missions were initially budgeted for an expected 90 days of operation in 2004. However, nearly two years later, both rovers are still running. This extra long run has cost NASA $80 million more than expected, mainly for operational costs [17][1]. This means that it costs NASA approximately **$100,000 per day** in salaries to monitor just two robots. Also, over the course of two years the rovers have traversed less than 10 square miles combined [16]. From another point of view, the overall cost per square mile is in the millions. Thus, the manpower cost in conducting robot operations can also add up significantly.

In all, Spirit and Opportunity require a team of several hundred people, including a crew of dozens to monitor and control these robots around the clock [2]. It is evident that lowering the human-to-robot ratio would decrease exploration costs. Future human-multirobot space exploration will likely involve humans and robots in close enough proximity to enable more direct telesupervisory or teleoperative intervention on an as-needed basis [2].

Though it may seem like an great idea to make robots fully autonomous, in practice it is difficult and rather rare. The main limiting factors are the reliability of autonomy and the intelligence of robots. Full robotic autonomy in any but the simplest tasks is still a long way off, and humans excel in complex or unexpected situations. Thus, human telesupervision of multirobot teams is an approach that combines human intelligence with robots' mechanical robustness and durability.

As robots become more autonomous and reliable, the number of robots a single human can

oversee in a group will increase since they will require less attention. This decrease in the human-to-robot ratio can potentially decrease the time it takes to accomplish a mission. However, with more robots on a team, a human supervisor's attention becomes increasingly divided. If many robots require human attention simultaneously, it is useful for these tasks to be prioritized, or scheduled, for the human.

Most scheduling problems in real-world applications are unique in that they are highly task-dependent, as well as objective-driven. One particular task characteristic of scheduling for human multirobot supervisory control is that human tasks become precedence constraints for the robots' remaining tasks—the robot must wait until the human is finished with it before regaining autonomy and returning to its own tasks. The time during which the robot is waiting for the human is known as downtime. One example arises when a robot fails and must wait until the operator fixes it. Another example is when a robot needs human intervention to interpret a situation, such as teleoperation for navigational guidance, or use of a robot as a remote science tool for experts.

Since downtime means that robots must wait for the human, scheduling for the human to lower downtime results in robots regaining autonomy sooner. We have developed an on-line, non-preemptive, heuristic scheduling algorithm to reduce downtime. This quick and efficient scheduling algorithm, dubbed *double Shifted Shortest Processing Time* (dSSPT), is ideal for human supervisory control. It can be used to prioritize tasks for the human supervisor such that the robot team performs its mission more efficiently, with more robot tasks performed sooner.

Having more robots on the team may also save money by allowing more tasks to be performed sooner if robots can share tasks. For example, in an area coverage problem, two robots can cover an area faster than just one. However, since human tasks are a precedence constraint for robot tasks, the time it takes for the entire mission (makespan) can limited by how quickly the human can address robot requests. When a human has no idle time left between any of his tasks, he is *saturated*, which means additional robots will not lower the makespan any further since the human cannot address those tasks any faster. The *saturation point* (also known as *span-of-control*) is also indicative of the maximum number of robots a human should handle since no further time can be saved from adding more robots.

Previous work by Goodrich and Olsen [13] has provided an estimate for saturation (known as *fan-out*) based on offline averages of interaction and neglect times between human and robots. This paper shows that the fan-out averaging technique does not capture variances observed in real

problems, and thus the actual saturation point is larger than an average estimate for most problems. The scheduling algorithm used for prioritizing human tasks also affects the saturation point on a team. A comparison is made between the scheduling algorithms compared (including dSSPT) and the fan-out estimate for simulated missions.

One contribution of this research is to present an online scheduling algorithm, dSSPT, which can more efficiently utilize a human's time. dSSPT prioritizes tasks for the human supervisor such that more robot tasks are completed sooner, as shown in the experiments described below. We also find that the team size saturation point increases with the variance of task length, as well as randomness (or aperiodicity) of occurrence. The scheduling algorithm used also influences saturation point. These findings deviate from the saturation proposed by Goodrich and Olsen's fan-out (Equation 2.1), which is based on an average estimate. Another significant contribution of this research is the analysis of reasons behind this discrepancy and the factors which affect team saturation.

Chapter 2

Related Work

This body of research builds upon work from various fields including:

- 2.1 Scehduling in Supervisory Control
- 2.2 Scheduling in Human-Multirobot Supervisory Control
- 2.3 Classical Scheduling
- 2.4 Maximium Team Size Issue in Human Multirobot Supervisory Control

2.1 Supervisory Control—What it is, how is it relevant?

Supervisory control describes systems where humans are continuously monitoring and manipulating the tasks being performed by machines. This type of control is also refered to as Human-in-the-Loop since the operator becomes feedback for otherwise open-loop systems, and also as supervised autonomous systems [8]. In the early 1960s, researchers became interested in manual control of automation—where the human operator's function was to specify parameters for a machine's automated action. The term supervisory control was first coined by Ferrell and Sheridan 1967 [9] as part of research into how humans can teleoperate vehicles on the moon given time delays. They realized that teleoperation in the conventional move-and-wait sense was very tedious and proposed that operators should give higher level instructions while machines perform this task autonomously.

In the robotics community, the ideal of full robotic autonomy for anything beyond very specific tasks is still a major challenge. Most real-world application of intelligent robotics (excluding in-

dustrial) have some form of supervisory control. For example, mobile robots are increasingly used in environments that are dangerous to humans—space, disaster zones, underwater, air. In these complex environments, the benefit of having a human supervisor is obvious—a higher cognitive intelligence to handle situations difficult for computer AI to interpret.

This dynamic between humans and robots has sprung the topic of human-robot interaction (HRI) that has been gaining momentum since the early 1990's [3]. This area of research considers the design of robotic systems with human factor considerations (looking at a system from a human user's perspective). HRI topics range widely from social interaction to physical form to cognitive load. This thesis will discuss mainly HRI relevant to supervisory control of multi-robot teams.

Scheduling in Supervisory Control

In the industrial and manufacturing domain, supervisory control has a longer history. MacCarthy et al. [12] list the history of significant works contributing to human factors studies in planning and scheduling since the 1960's, but conclude that knowledge in this field is still lacking and as a result, no methodical process of designing an effective supervisory control system exists. The research in this document does not attempt to solve this complex issue in general, rather, it focuses on a sub-problem of human-multirobot supervisory control and examines human factors in scheduling relating to multirobot metrics and attempts to generalize some observations about efficient use of human time.

2.2 Scheduling in Human-Multirobot Supervisory Control

Currently, most practical applications of robot teams have a very small robot-to-human ratio. For example, the state of the art in robotic space exploration is the Mars Exploration Rover (MER), which requires a human crew of dozens to monitor and control just two robots[2]. The management of a multirobot team where one operator can oversee many robots is very much a developing topic of research.

In a related study, Cummings and Mitchell [7] spell out various management issues in the control of multiple UAVs. They find that humans are not good at selecting the best course of action when there are many complex possibilities, even when given a preview of the upcoming timeline. Also, given an increased workload, the amount of time humans plan ahead decreases, resulting in

less efficient plans. This indicates that humans are less reliable when it comes to managing larger groups of robots, which suggests more guidance (e.g., scheduling) for humans may help.

Human Task Modelling

A scheduler assumes that the parameters of the tasks passed to it are accurate. For human-robot interaction, this means there must be a model for human behavior given certain tasks. Work has been done in this area by Goodrich and Boer [10] as well as his other students. This complex issue is not the focus of this report, however, scheduling with invalid paramters (for example, if we had a bad human model) would give no guarantees for a good schedule.

Human-Robot Interaction Metrics

The field of human-robot interaction is still relatively new. Recently, researchers working in this field have started suggesting common metrics to standardize research in the various facets of the issue. In particular, many prominent members of this area published some general metrics applicable (Steinfeld et al. [15]). This report will focus on a few metrics related to human-multirobot interaction.

One of the metrics suggested for tasks is *efficiency*, which in general measures the time needed to complete the task. Efficiency measures can include:

- Time to complete the task
- Operator time for the task (includes HRI overhead)
- Average time for obstacle extraction

The research described in this report will look at efficiency in terms of operator time as well as the time and rate of mission completion for the entire team (human and robots).

Another suggested metric for management of multirobot teams is fan-out. Fan-out, as defined in Goodrich and Olsen [13], is a measure of how many robots can be effectively controlled by a human. This report will break this metric down further to determine the factors which affect team size.

A note on term definitions, the terms *fan-out* and *span-of-control* descriptively have the same definition. In all the literature that describe fan-out, Equation 2.1 (which supposedly defines the

upper bound of fan-out) is always associated with the term. However, we find out though experiment that the equation is not always true, thus this report will use the term *span-of-control to avoid* confusion. *Span-of-control* is a more general term that is used beyond the area of human-multirobot interaction, notably in business management and economics.

2.3 Classical Scheduling

Scheduling theory has been very well studied for over half a century. The classic scheduling problem is to order a set of tasks such that a certain objective is achieved (usually something is minimized or maximized).

Definitions

- *Release Time (r_i)*—The time at which task i arrives to be scheduled.
- *Processing Time (a_i)*—The expected time required to complete task i.
- *Interaction Time (IT_i)*—Another phrase for processing time, but used in the human-multirobot application domain.
- *Resource*—The entity that will process the task.
- *Idle Time*—Time during which the resource is not processing a task.
- *Completion Time or Makespan*—Time it takes to complete all tasks in a schedule.
- *Wait-time*—The time that a task has to wait before being addressed by the resource.
- *Downtime or Flow Time*—The time a task spends in the system, i.e., the wait-time plus the processing time of a task.
- *Over-subscribed*—Having multiple tasks queued.
- *(*Precedence constraint)—A succession relationship between tasks.
- *Preemption*—The idea that a task can be interrupted during processing. This task may, or may not be resumable. There are generally three categories when it comes to preemptability of tasks:

1. *preemptive*—A task can be interrupted during processing, placed on hold, and then resumed from the point of interruption. It can also describe a task that can be broken into smaller chunks and scheduled at various times, not consecutively.
2. *semi-preemptive*—A task that is interruptable, but not resumable. That is, any incomplete work must be restarted. For example, tasks with a high cost for context switching [11].
3. *non-preemptive*—A task is not interruptable, and thus not resumable either.

- *On-line vs. off-line*—Off-line scheduling occurs when the complete set of tasks and its parameters are known ahead of time. On-line scheduling occurs when there is no prior knowledge of new task arrivals until they are released. The decisions are made based only on the known tasks.

- *Dynamic vs. static*—Static means not time-varying processing times, for tasks and dynamic means processing times for tasks can be different or vary over time. (Note: There seems to be some differences in definition of static and dynamic amongst scheduling literature. Some define static scheduling as requiring a complete knowledge of the task set and its constraints for the scheduling task. All scheduling decisions are made during compilation and that schedule is fixed for all time [14]. This seems to be the same definition as an off-line algorithm, so we will not use this definition.)

Classical Scheduling Algorithms

Some of the most common algorithms are First-In-First-Out (FIFO), which orders tasks based on when they arrive; and Shortest Processing Time (SPT), which orders based on processing time to minimize waiting times for other tasks. They are widely used due to their simplicity, speed and performance. For all scheduling algorithms, it is assumed that each task provides a complete set of relevant information, including release time and processing time of the task. These algorithms are explained in greater detail below.

Although the algorithms described in this section are simple and fast, they are unfortunately all sub-optimal for minimizing downtime (and thus can, and will be improved upon). Another point of note is that most scheduling problems in real-world applications are unique and highly

task-dependent, so many scheduling algorithms are customized for a particular need. Thus, for our particular focus on human-multirobot group telesupervisory control, we seek to minimize overall downtime by using an on-line, non-preemptive scheduling algorithm. Further reasoning and assumptions are described in the next section along with a derivation of a new scheduling algorithm which outperforms those described in this section.

Shortest Processing Time (SPT) and its derivatives (including SRPT and SSPT)

One classical scheduling algorithm is the Shortest Processing Time algorithm, which schedules the tasks by non-decreasing processing times. This can be expressed as $a_1 \leq a_2 \leq \cdots \leq a_n$. If all tasks arrive simultaneously, this algorithm minimizes the overall completion time, the mean waiting time and the maximum waiting time. However, this ideal condition of tasks arriving simultaneously obviously does not hold on-line. Thus, it performs sub-optimally for minimizing wait-time and downtime in real-life applications.

A variation of SPT for jobs that arrive at different times is known as Shifted SPT (SSPT). It can be expressed as $r_1 + a_1 \leq r_2 + a_2 \leq \cdots \leq r_n + a_n$. where the subscripts sequentially designate the re-ordered task order, rather than the release-time order. Like SPT, this algorithm orders by task end-time, which is now found by adding the release time. The objective is to develop a feasible on-line heuristic which attempts to minimize the sum of completion times over all tasks [8]. It is also a sub-optimal algorithm, but was proven to have a 2-competitiveness (it performs at most 2 times slower than the offline optimal algorithm) for a single-resource system. Both SPT and SSPT are non-preemptive algorithms, but they have a preemptive analogue known as Shortest Remaining Processing Time (SRPT).

Shortest Remaining Processing Time algorithm (SRPT) is a preemptive algorithm which optimally minimizes total downtime. At each point in time, SRPT schedules the task with shortest remaining processing time, preempting when jobs of shorter processing times are introduced. The disadvantage of this greedy method is that long jobs may have to wait a long time. It is also a preemptive algorithm, and because this paper's focus is on semi-preemptive algorithms, SRPT was not used.

First-In-First-Out (FIFO) or First-Come-First-Serve

This is the most commonly used scheduling algorithm, also known as a queue. This greedy algorithm schedules tasks in the order they arrive (in other words, in order of release time.) This can be expressed as $r_1 \leq r_2 \leq \cdots \leq r_n$. The advantage of this algorithm is that the implementation is very simple: a queue. It also minimizes maximum downtime [4]. The disadvantage is that the waiting time for a task depends on its time of arrival, so short tasks may be stuck waiting for a long time for long tasks to finish, resulting in poor average performance. Thus, when there are large variations in processing times of tasks, FIFO is very much sub-optimal in terms of minimizing downtime.

Objective of Downtime

Downtime, also known as flow-time, is defined as the time a task is released until the time it is completed. It is a metric that is closely related to the task's source's experience (i.e. robot) since it measures the amount of time it will have to wait to get its task serviced. Other works have looked into this metric, in particular, Bansal 2003 [4] that looked at minimizing related downtime metrics such as p-norms mainly for preepmtive tasks. Our work focuses on minimizing total downtime for semi-preemptive tasks. More details about downtime in relation to human-multirobot interaction is described in the next chapter (Chapter 3).

2.4 The Issue of Team Size in Human-Multiobot Interaction

One important metric in human multirobot interaction (HMRI) is span-of-control, which denotes the number of robots a human can supervise. The related term *fan-out (FO)*, coined by Goodrich and Olsen [13], proposes an upper bound on team size for supervised multirobot teams as:

$$FO \leq \frac{NT}{IT} + 1 \tag{2.1}$$

where *IT (interaction time)* is the time it takes on average for humans to address the robot's request and *NT (neglect time)* is the average amount of time robots can act autonomously without human intervention.

This bound given by Equation 2.1 is based on two explicit assumptions: all the robots on the team are homogeneous (Crandall later extends the idea to a feasibility test for heterogeneous teams

Figure 2.1: Illustration of team size can be estimated by the number of ITs that can "fit" in NT, plus 1 (from Crandall [6])

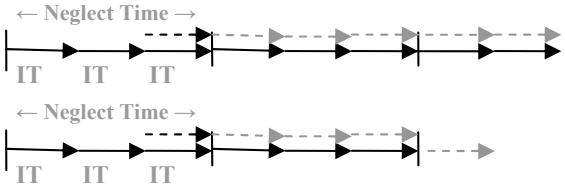


Figure 2.2: Top shows 4 tasks arriving in an NT that can only fit 3 so one task never gets addressed if the timeline is infinite. Bottom shows the 4th task addressed at the end for a finite timeline.

[6]), and average NT and IT values yield a valid fan-out value. This suggested upper bound can also be interpreted as: A team should be of a size such that, on average, only one robot will require human attention at any given time.

The rationale behind it is that, since humans can only attend to one robot at a time, having more robots requiring help simultaneously means that some will not be addressed; in other words, the human is saturated. However, in any practical applications, the NT and IT values will vary from the estimated mean and requests will not be as ideally periodic as in Figure 2.1. Therefore, as team size approaches the upper bound in Equation 2.1, the probability of more than one robot requiring assistance at the same time (over-subscription) is very high. In the worst case, all robots could require human attention simultaneously, despite the fact that the average fan-out is within the bounds of Equation 2.1. Those robots waiting for human attention experience downtime.

Another, implicit, assumption is that the averaging takes place over an infinite time horizon. The upper bound proposed by Goodrich and Olsen is true on an infinite time horizon, where having more ITs than can "fit" into NT will result in tasks not being addressed. For example, if one extra IT task occurs, on the infinite time horizon one task will always be left unfinished (Figure 2.2 top). However, most robotic missions have a bounded time horizon and set goals. It is also impossible for humans to work indefinitely long, and even if there are many humans rotating, funding for the mission is surely limited. When a mission ends, the unfinished task in the previous example will eventually be addressed, with the difference being that the makespan, or time to complete all tasks,

becomes IT units of time later than without that task (Figure 2.2 bottom).

The research described in this report considers what happens to the span-of-control when variance in IT and NT occur (as would be expected in realistic applications) and how we can increase span-of-control even further by using a smart scheduling algorithm.

Chapter 3

Approach

This section outlines the derivation of the dSSPT algorithm which has the objective of lowering total downtime for the robot group. The examples and explanations given assume that each task is coming from individual robots to a supervisory control list used to monitor tasks and schedule them to optimize human-multirobot group efficiency in terms of downtime.

3.1 Downtime as a Human-Multirobot Scheduling Objective

Most scheduling problems in real-world applications are unique. Factors that influence which scheduling algorithm to use includes the mission objective, the types of tasks being scheduled and the constraints on those tasks. Interestingly, human tasks in supervisory control all have one characteristic in common:

Human tasks become precedence constraints for the robots' remaining tasks.

In other words, the robot must wait until the human is finished with it before regaining autonomy and returning to its own tasks. The time during which the robot is waiting for the human is known as downtime as defined earlier.

Since one popular reason for using multiple robots is so that they can do more work autonomously, one natural objective to having when scheduling human tasks would be to minimize robot downtime such that robots can return to autonomy sooner. This is the objective that the following derived algorithm dSSPT is based on. Since the goal is to use it on real systems, the scheduling algorithm must be online.

There are generally three types of tasks in scheduling as mentioned in the previous section: preemptive (interruptable and resumable), semi-preemptive (interruptable and not resumable) and non-preemptive (not interruptable or resumable). For preemptive tasks, an existing algorithm of Shortest Remaining Processing Time (SRPT) is already known to minimize downtime. For non-preemptive tasks, if the first task in a task queue is always held constant (due to no interruptions to task being addressed), then the arrival times of all the remaining tasks in the queue does not matter. Thus those tasks can be scheduled in Shortest Processing Time (SPT) order to minimize downtime for that task queue instance. Section 3.3 in the derivation below will illustrate why that is the case. For semi-preemptive tasks, the first task in the queue can be switched as well and there is currently no online heuristic algorithm that minimizes downtime when tasks arrive asynchronously. The dSSPT algorithm derived below will address this problem of semi-preemptive tasks and turns out to outperform SPT and other algorithms. dSSPT also ties SPT for downtime for non-preemptive tasks.

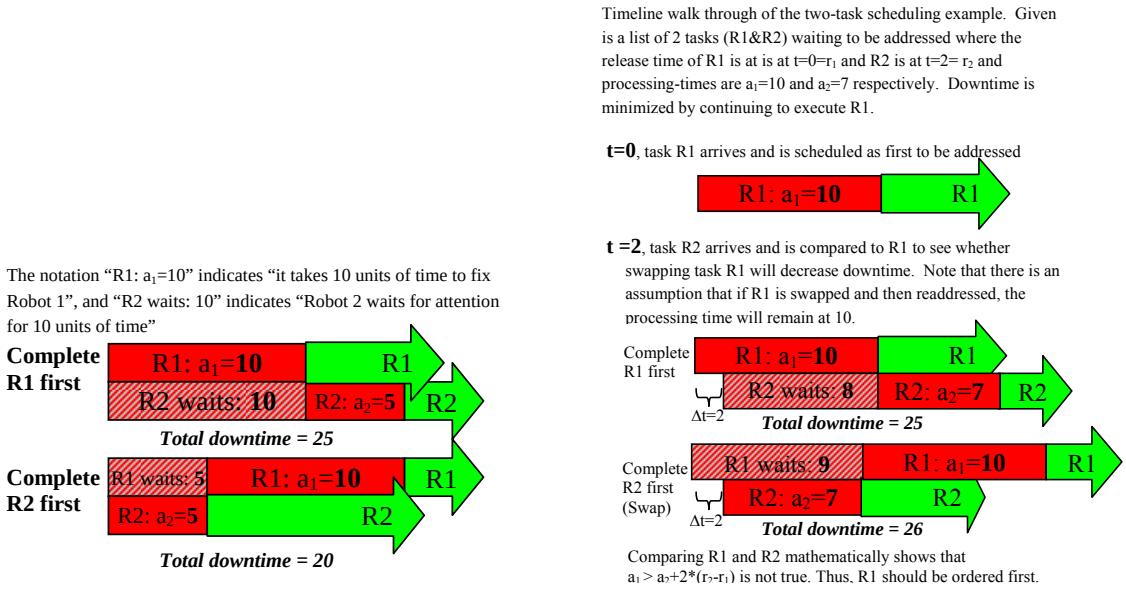
3.2 Assumptions

Since a human overseeing a group of robots cannot possibly be aware of every robot situation simultaneously, it is assumed in simultation that he will take the advice of the supervisory control list to provide him with an efficient order in which to attend to the robot tasks.

All the task parameters, such as processing time and release time, are assumed to be known or accurately estimated by the robot. All tasks are equal in priority and each task is independent of the others. In practice, it is often not the case that each task has equal priority, so weighted tasks may be considered in future work on this algorithm. The type of tasks assumed by this algorithm are semi-preemptive, instantaneously static with time, which is further explained in the next subsection, and without deadline. The cost of context switching is assumed to be embedded in the processing time of the task.

3.3 Derviation of dSSPT

For the basic case of two tasks arriving simultaneously, we can address task 1 before task 2 or the other way around. This is illustrated in Figure 3.1a. An example of the basic case is given to



(a) Simultaneous robot tasks to be ordered.

(b) Timeline of scheduling example.

Figure 3.1: Two robot scheduling examples.

demonstrate the effect of scheduling order. Imagine two robots requiring assistance sending their tasks simultaneously to the human supervisor. Robot 1 requires 10 units of time to fix and Robot 2 requires 5 units. Assuming that the performance of Robot 1 is independent of that of Robot 2, ordering robots by Shortest Processing Time minimizes total downtime for the robot team as seen in Figure 2. In the diagrams, ai indicates the processing time of task i.

When we incorporate the idea that tasks may arrive at different times, then the time delay between task arrivals, Δt, has to be taken into account (Figure 3.1b). When the first task arrives, it is immediately addressed. However, for the two-robot case, when a new task arrives before the completion of the first task, the problem becomes whether the downtime can be decreased by interrupting the first task to address the new task first. We can see that if we do not swap, the total downtime will be $2a_1 - \Delta t + a_2$ whereas if we do swap, downtime would be $2a_2 + \Delta t + a_1$, where $\Delta t = r_2 - r_1$ and r_i is the release time for task i. Comparing the two equations for minimal downtime, the criterion used for the swapping decision is thus $a_2 + 2\Delta t < a_1$. A timeline of events is illustrated in Figure 3.1b. If the processing time of the new task from Robot 2 is shorter than the processing time of Robot 1 by a difference of more than $2\Delta t$, then swapping results in less downtime. This example suggests that downtime can be minimized by ordering shorter tasks before longer tasks

with the time delay factored in. The main comparative equation for deciding whether to swap is $a_1 > a_2 + 2(r_2 - r_1)$ or $a_1 + 2r_1 > a_2 + 2r_2$ where r_i is the release time for task i.

We can see that this on-line algorithm being developed takes a look at the snapshot of the set of tasks and its parameters at a particular moment in time, thus running it continuously makes it a dynamic scheduling algorithm. In these simple examples used for illustration, the processing times remain static; however, in real-life, these estimated processing times will be varying. Therefore, in real-life, the processing times should be updated continuously.

The timeline in Figure 3.1b also illustrates the semi-preemptive assumption—that if the processing of R1 is interrupted, when it is later readdressed the processing time is assumed to still be 10 as opposed to 8. This implies an assumption about the type of tasks being scheduled. If this algorithm is being used on-line it means a task is assumed to degrade or return back to its original state if left unattended, and thus cannot be broken up into further chunks for scheduling. Such tasks might include exploration sciences, such as picking up soil samples or taking a measurement of the environment. This operator-determined task interruption is not to be confused with pre-emptive scheduling. This algorithm is not pre-emptive in the conventional sense in that it will not break up tasks into chunks to schedule (unlike SRPT). This distinction will become clearer when dealing with more robots.

It is also assumed that when running on-line in a dynamic situation where each task's values are updated continuously, the parameters would reflect the actual state of the tasks (i.e., if the 2 units of time spent attending to R1 reduced a1 from 10 to 8, then at time $t = 2$, a_1 would have been updated to reflect the 8.) If there a reduction in processing time then there will be less incentive to swap to R2, which is a feature of this algorithm that leans towards continual work on a task that is progressing quickly.

The idea of factoring in the release time resembles the SSPT application that orders tasks such that the processing times for each task is "shifted" by its release time: $r_1 + a_1 < r_2 + a_2 \Rightarrow a_1 < a_2 + (r_2 - r_1)$ or $a_2 + \Delta t$ [4], which is also considered an on-line algorithm. The factor of 2 in our algorithm turns out to be of major significance in minimizing downtime. Due to this factor, we named our algorithm "double Shifted SPT" (dSSPT). This train of thought will be extended in the next section to many tasks instead of just two.

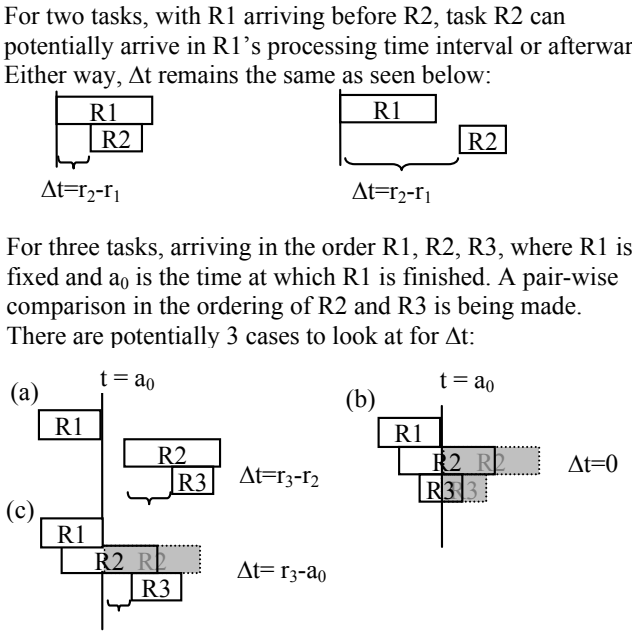


Figure 3.2: Time shift for 3 robots

Extending it to N Robots

We have already shown that in the two-task case we can minimize the downtime of robot groups by ordering them by a time-adjusted SPT. Extending our two-task example to three tasks, we wish to determine whether a similar pattern exists.

With three tasks, the Δt becomes more complex, as graphically depicted in Figure 3.2. Again, taking the example of tasks coming from individual robots, for a two-robot comparison, Δt is simply a chunk of time used to represent the difference in time between when the two tasks arrive. In other words, it can simply be expressed as the difference in release times of the tasks: $\Delta t = r_2 - r_1$. The bottom half of Figure 3.2 illustrates the other possible cases when three tasks are involved. Since the scheduling algorithm is making its ordering consideration based on a snapshot of the current situation, in the case of the three tasks the decision for Δt is made when R3 is released ($t = r_j$) as given in Figure 3.2. When running this on-line, the cases of 3.2(a) and (c) would mean there are only two tasks remaining in the list (R2 and R3).

We can see that the SPT principle also applies for three robots but there is an added complication when tasks (R2 and/or R3) arrive before the previous task finishes (R1). In which case, the task(s) that arrive(s) earlier will be forced to wait until R1 finishes. Since there is no choice about the

matter, that wait-time is not factored into our dSSPT ordering equation. Thus, the Δt is different depending on the situation.

Looking at cases 3.2(a) and (c), both with two tasks in list, we can see that it is very similar to the two-task problem described above. The major difference from an off-line point of view is that in (c) task R2 is waiting for R1 to finish before it can start. Thus, cases (a) and (c) can be generalized as one case where Δt can be described as the chunk of time between the start time of R2 and the release time of R3. This makes the Δt for (a) and (c) become just one case when running on-line since R1 will not be in the list, thus when R3 arrives, it will not know that R2 had to wait for $R1 - a_0$ is implicitly obtained. However, for the purpose of clarity we will make this case distinction explicit.

What happens when we apply this equation to case (b)? The projected start time of R2 is a_0, which is later than the release time of R3. This means r_3-(start time of R2) will be negative. In other words, both tasks R2 and R3 must wait for R1 to finish before they decide which of them goes first. In such a case, a SPT approach makes sense to minimize downtime.

To verify this mathematically, we look at the three different cases for determining Δt: Assume that task R2 is released before R3 ($r_2 < r_3$) and a_0 is the time at which task R1 is completed. Let the time difference of tasks R2 and R3 relative to task R1 be: $\Delta t_2 = r_2 - r_1$ and $\Delta t_3 = r_3 - r_1$ Let tasks processing times for tasks R1, R2 and R3 be a_1, a_2 and a_3 respectively. Let $t = a_0$ be the time at which R1 is completed.

1. If $a_0 > r_2$ AND $a_0 > r_3$ (see Figure 3.2(b)) The downtime for R2 before R3 is:

$$(a_1 - \Delta t_2 + a_2) + (a_1 - \Delta t_3 + a_2 + a_3) \tag{3.1a}$$

   The downtime for R3 before R2 is:

$$(a_1 - \Delta t_3 + a_3) + (a_1 - \Delta t_2 + a_2 + a_3) \tag{3.1b}$$

   We want to swap R3 ahead of R2 if its downtime is smaller. Setting $3.1b < 3.1a$, we end up with the simple comparison $a_3 < a_2$. Thus, giving us $\Delta t = 0$.

2. If $a_0 > r_2$ AND $a_0 < r_3$ (see Figure 3.2(c)) The downtime for R2 before R3 is:

$$(a1 - \Delta t2 + a2) + (a2 + a3 - (r3 - a0)) \tag{3.2a}$$

The downtime for R3 before R2 is:

$$a_3 + (a_1 - \Delta t_2 + (r_3 - a_0) + a_2 + a_3) \tag{3.2b}$$

We want to swap R3 a R2 if its downtime is smaller. Setting 3.2*b* < 3.2*a*, we end up with the comparison of $a_3 + (r_3 - a_0) < a_2 - (r_3 - a_0)$. Thus, giving us $\Delta t = r_3 - a_0$.

3. If $a_0 < r_2$ AND $a_0 < r_3$ (see Figure 3.2(a)) This is the same as two-robot case described in the previous section, giving us $\Delta t = r_3 - r_2$.

As mentioned before, Δt can be generically expressed as r_3−start time of R2 , where start time of $R2 = \max(a_0, r_2)$ as can be seen in Figure 3.2.

We will now further derive the effect of release times for n tasks. Following the logic in the three-task example, if there are more than two tasks waiting in the list simultaneously, all subsequent tasks from number two onward have to wait until the completion of the first task. Thus any tasks in positions two and up in the list will have the lowest downtime by ordering the list using SPT (set $\Delta t = 0$) If there are two robots in the list simultaneously, then the effect of the difference between the release time of the second task and the start time of the first task is considered to determine which order lowers downtime.

Although this may seem like a small difference from SPT, when running on-line, this two-task case comes up quite often and dSSPT can actually make a significant improvement over SPT. This will be demonstrated in the experiments.

3.4 dSSPT Algorithm

Each new task is sorted into the dSSPT-scheduled list by comparing the new task to existing tasks starting from the end of the list and working towards the beginning. Given a new task j, and an existing task in the list i (with $r_i \leq r_j$), do pair-wise comparisons down the list according to the following:

Δt can also be expressed as $\Delta t = r_3$−start time of R2, where the start time of $R2 = \max(a_0, r_2)$.

If $a_j + 2\Delta t < a_i$ is true,

where $\Delta t = \begin{cases} 0 & , \text{ if i} \geq 2 \\ r_j - max(a_0, r_i) & , \text{otherwise} \end{cases}$

then swap tasks i and j.

Do this for tasks $i = j-1, j-2, ..., 2, 1$

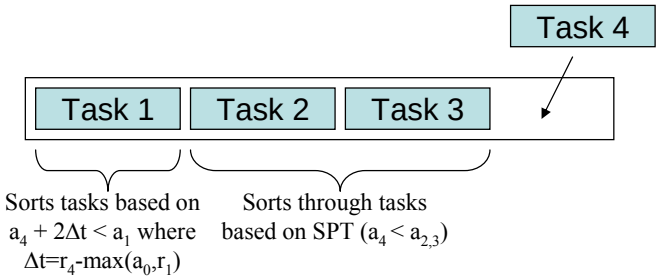


Figure 3.3: An example of a new task arriving and sorted into the task queue. If the new task is the smallest in the queue it will be sorted to the second position, but not necessarily the first position (depending on release and start times.)

3.5 dSSPT Compared to SPT Variants

dSSPT is essentially a hybrid of an improved Shifted-SPT and the original SPT in the different regimes where those algorithms perform best. For sparse schedules, it uses the improved Shifted-SPT and for dense schedules (three or more tasks queued), it uses SPT for the latter tasks. Figure 3.5 illustrates when a new task how it gets sorted through the list based on different criteria. Thus, the drawbacks of dSSPT for dense schedules are like those of SPT in that longer tasks may have to wait for a very long time if there are many tasks queued.

3.6 Bounds and Complexity

Computational Complexity

The computation time for this greedy scheduling algorithm has the same complexity as other classic algorithms such as SPT, and SSPT of $\omega(n)$, thus making dSSPT an efficient polynomial time alternative to those algorithms.

Competitiveness

It is infrequent to have an online algorithm that is the same as the offline optimal. For offline scheduling, all tasks are known apriori whereas online algorithms only know of the existance of a task when they arrive, thus the decisions are made based only on partial knowledge of all possible tasks. Online is ofcourse the most realistic and suitable for real applications.

For competitiveness, an online algorithm is compared to the offline optimal solution as a ratio. In terms of downtime, dSSPT should never perform worse than SPT for any situation. Since the first task in the queue is held constant (not preemptable), all sequent tasks are waiting for that first task. As shown in Section 3.3 above, for those tasks waiting in the queue the order in which they were released no longer matters and SPT is ideal. Since dSSPT schedules the same as SPT for these dense queues, dSSPT has the same competitiveness as SPT of $(\delta+1)/2$-competitive for downtime of non-preemptive tasks [5], where δ is the ratio between the longest and shortest job lengths. Bunde also shows that $(\delta+1)/2$-competitive for downtime is the best possible for any deterministic algorithm [5].

For both non-preemptive and semi-preemptive tasks, the offline optimal solution is the same since the tasks cannot be broken into smaller tasks (i.e.,not resumable). Thus the competitive results of semi-preemptive tasks can be directly related to non-preemptive tasks. For semi-preemptive tasks where the release times of the first two tasks in queue have an impact on downtime if switched, we know that SPT performs poorly for this case because it does not factor in release time at all. So a short task that comes last will be sorted to the front of the list, thus making all tasks have to wait both the difference in the release times of the first task and last task as well as the processing time. This is clearly sub-optimal and intuitively should have a competitive ratio greater than or equal to $(\delta+1)/2$ (the math is pending).

However, it can be logically reasoned that dSSPT scheduling of semi-preemptive tasks will result in lower or equal downtime to SPT scheduling of non-preemptive tasks for every instance of the schedule: If there is only one task in the queue when a new task is added dSSPT guarantees minimal downtime whereas SPT does not due to not incorporating difference in release times. If there are many tasks in queue, since the heuristic does not resort all tasks after the placement of the new task, there is potential that swapping with the first task will improve something locally may affect something globally. I have derived no guarantees mathematically for global improvement

over any other algorithm, however, global improvement on average can be seen in the experimental results in the next section. Thus, further analysis is still necessary to rigiorously prove the intuition above.

3.7 dSSPT Extensions

This section discusses how to extend dSSPT to incorporate other aspects of scheduling.

Weights

Weights (w_i) are a common method to assign an explicit priority on tasks. Usually, the processing time of a task would be weighted by a constant associated with that task. A weighted SPT comparison for a two-task swapping problem would be evaluated based on:

$$w_j a_j > w_i a_i \tag{3.3}$$

Extending to incorporate difference in start and release times:

$$w_j(a_j - 2\Delta t) > w_i a_i \tag{3.4}$$

Cost of Context Switching

There is usually some overhead associated with interrupting tasks (context switching). The cost for this switch (denoted by C) may be constant or varying dependin on the situation. The comparison equation for switching would be modified to:

$$a_j - 2\Delta t > a_i + C \tag{3.5}$$

Notice that the cost effectively acts as a margin or debouncer to discourage swapping.

Using Human-Multirobot Interaction Terms

The processing time of a task can be thought of as the sum of the interaction and neglect time:

$$a_i = IT_i + NT_i \tag{3.6}$$

Crandall and Cummings [6] and Elfes et al. [8] extend break down the models further to include wait time (the period of time between when a task is released and when it is addressed).

Chapter 4

Characterization of Algorithm

4.1 Uni-Resource Scheduling Problem

Downtime Efficiency

The experiments outlined here are designed with a single human as a resource for a multirobot group in mind. Since there has been little work done in terms of scheduling for human-multi robot groups, we mainly compare our scheduling algorithm to FIFO as a baseline. We also compare dSSPT to its precursors, namely SPT and SSPT. Each experimental data point presented below was obtained by averaging over 100 trials.

In one set of experiments, the IT of tasks are drawn from a Gaussian distribution with a mean of 15 units of time. The variance was changed for each set of runs from 1 to 6. Each robot was assumed to have a NT of 180 units, thus limiting each robot to reporting a task only once within that time span at a time chosen randomly with uniform probability across the span. Figure 4.1 plots the downtime for an IT variance of 1 and 6, experienced on average by varying robot group sizes for each scheduling algorithm. For both trials, FIFO and SSPT perform the same on average with SPT performing similarly as well while dSSPT outperforms them all.

For the trial given in Figure 4.1(a) with a mean IT of 15 and variance of 1, the dSSPT schedule had a downtime improvement over FIFO that increases proportional to the robot group size and plateaus at around 30% for robot groups of 17 or larger. This percentage improvement of dSSPT over FIFO is shown in Figure 4.1 for the trial plotted in Figure 4.1(c). The shape of this percentage improvement curve is similar for all experiment comparisons, so the other plots were not reproduced

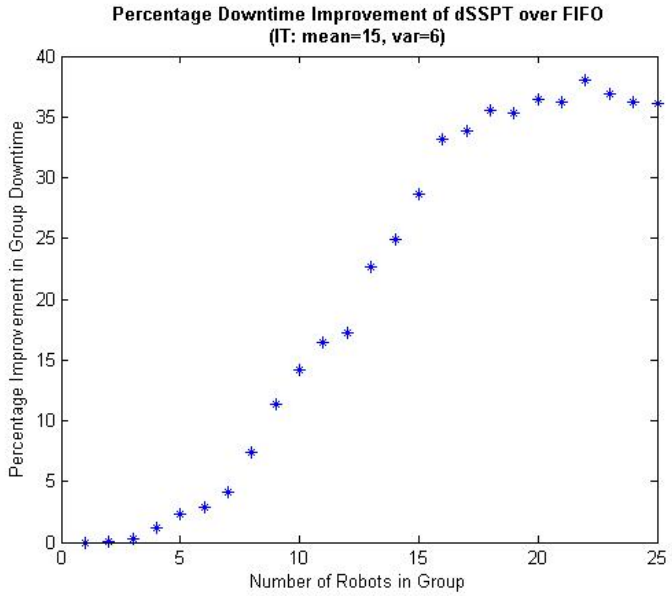


Figure 4.1: Percentage Downtime Improvement of dSSPT over FIFO.

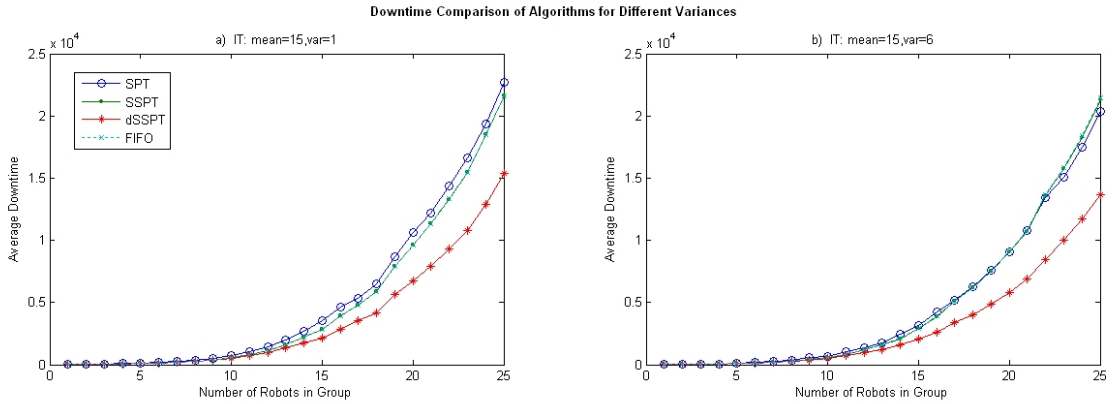


Figure 4.2: Average downtime comparison for increasing team size and changing task variances.

here. In Figure 4.1(b), where the *IT* had a variance of 6, the dSSPT schedule had a downtime improvement over FIFO that plateaus around 36% for groups of 19 and above. We note that with more variance, dSSPT's performance increases and so does SPT's. Also, the point at which the downtime improvement over FIFO plateaus is correlated to the maximum number of robots that can be scheduled within the *NT*. A more detailed analysis of the number of robots addressed within *NT* is given in the next section.

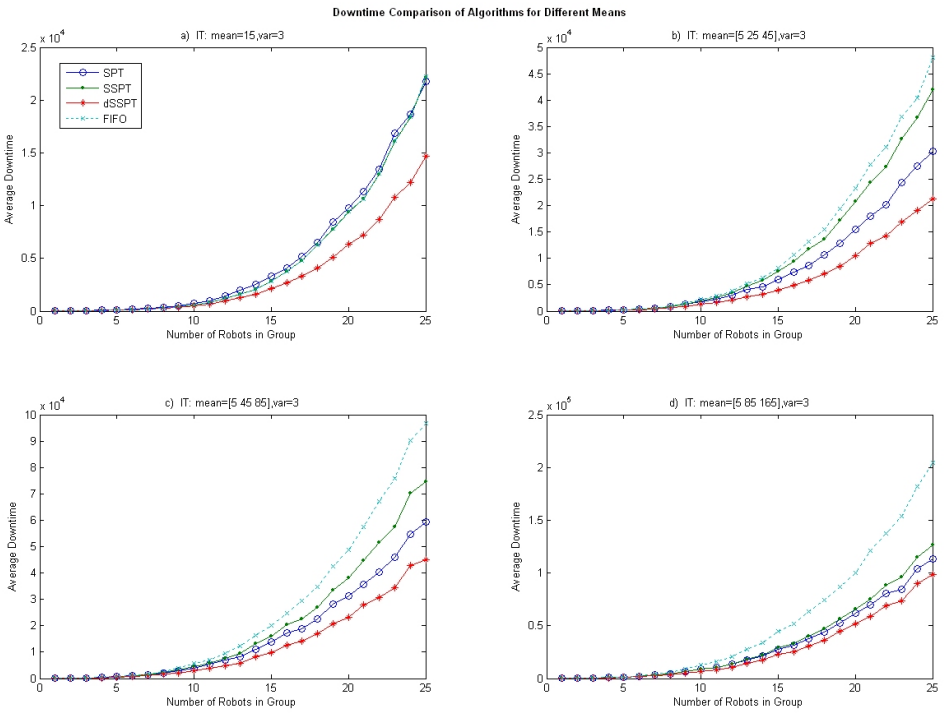


Figure 4.3: Average downtime comparison for increasing team size and randomly drawn task mean.

It makes intuitive sense that FIFO ($r_1 \leq r_2 \leq \cdots \leq r_n$) and SSPT ($r_1 + a_1 \leq r_2 + a_2 \leq \cdots \leq r_n + a_n$) performed nearly the same on average if the mean IT $(a_1, a_2, \ldots, a_n)$ is the same on average. The SSPT basically delays every task release time by a constant offset: the mean fix-time ($a_1 \approx \cdots \approx a_n \approx const$). In other words, a SSPT schedule basically becomes a FIFO schedule that is offset by a constant. On average, the task ordering of the SSPT would be the same as FIFO. dSSPT's improvement over SSPT is due to two factors: 1) the factor of 2 used in dSSPT augments the differences in IT. Though IT is the same on average, the small variances are enhanced by a factor of two, distinguishing itself from FIFO and SSPT. 2) SSPT does not take into account the case where many tasks are consecutively delayed as shown in Case (c) of Figure 4. In this case, the release times become less significant than the start times.

In this set, the downtime from using FIFO, SSPT and dSSPT are not dramatically different (though dSSPT was definitely superior). FIFO's weakness arises as the lengths of task ITs vary more. This weakness can be more clearly seen in the second set of experiments given in Figure 4.1.

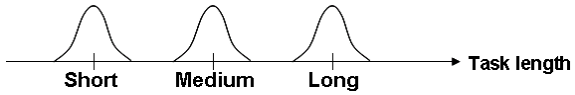


Figure 4.4: Random variable drawn from $\tilde{N}(mean, var)$ where $(mean, var)$ is drawn uniformly from a set of 3 possible lengths representing short, medium and long tasks. For example, $(5,3),(25,3),(45,3)$

The second set of experiments simulated tasks of three classes with short, medium and long IT (illustrated in Figure 4.1), which is a more realistic emulation of a real-world situation rather than having all tasks with nearly the same IT. This set of experiments drew an IT value from three possible normal distributions with means representing short, medium, and long ITs from [5, 25, and 45] to [5, 85, and 165] respectively, and a variance of 3, as illustrated in Figure 4.1.

FIFO starts performing progressively worse than SSPT for longer variations in IT. Also, SPT starts performing better for situations in which tasks are more densely scheduled, meaning there are more robots in the list simultaneously (over-subscribed). This shows that for a more sparsely populated task list, where breaks between tasks occur more often, release time is of crucial importance in scheduling, thus allowing dSSPT to perform better. When there are many simultaneously scheduled tasks, the IT lengths are more relevant, making SPT's performance improve. dSSPT was still superior to all other scheduling algorithms, since it adopts a time-shifted approach when tasks are sparse and the SPT approach when tasks are dense, thus adopting the best algorithm depending on the situation.

In Figure 4.1(b), dSSPT shows up to 54% improvement over FIFO, similarly in 4.1(c), it plateaus around 52%, and in 4.1(d) it is around 50%. In 4.1(d), the large tasks had a mean of 165 units of time, which is nearly spanning the entire neglect time of 180. When the ITs are all very large, it becomes difficult to schedule very many robots within neglect time span, thus explaining the declining performance of all algorithms as the ITs got larger.

The main insight from the equations and this set of experimental results is that dSSPT is basically a hybrid of an improved SSPT (with a factor of 2) and the original SPT in the different regimes where those algorithms perform best. Thus, the drawbacks of dSSPT for dense schedules are reminiscent of those for SPT, in that longer tasks may have to wait for a very long time if there are many tasks queued.

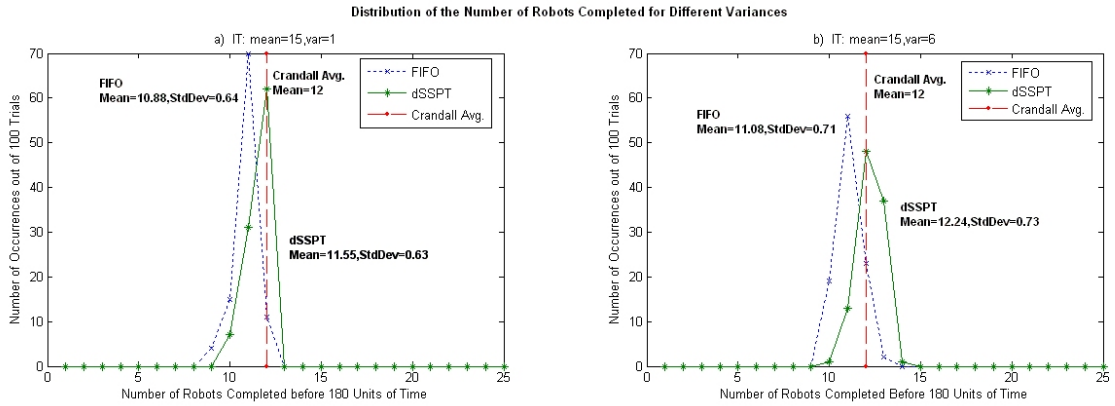


Figure 4.5: Number of tasks addressed within NT for a 25 robot team when changing task variance.

Human Tasks Performed in NT

As mentioned in related work, research by Olsen et al. predicts the span-of-control (the number of robots a human can efficiently handle) in human-multirobot groups. However, because their prediction method is based on static performance averages obtained through conducting experiments and simulation prior to the application itself, it tends to underestimate the efficiency obtained by dynamic on-line (and on-the-fly) scheduling. In this section we compare Crandall et al.'s span-of-control prediction based on averages with results obtained from dSSPT's dynamic, on-line scheduling. The major difference between these methods is particularly apparent when task ITs have larger variations, as would be expected based on the results of the previous experiments.

Using the same type of experiments described in the previous section, we placed 25 robots in the group, set the NT to 180 units of time and varied the IT variance (Figure 4.1) as well as mean to simulate long, medium and short tasks (Figure 4.1). For this section, the main focus is the number of robots that can be scheduled and completed within the span of NT.

According to the fan-out equation 2.1, each human operator can handle $NT/IT + 1$ robots since NT/IT robots can be addressed within the span of the (+1) robot's NT. For the first set of experiments where NT was 180, IT was 15 and variance varied from 1 to 6 (Figure 4.1), one would expect according to Goodrich, Olsen and Crandall that a $180/15 = 12$ robots can be addressed within the NT span. This was generally confirmed in simulation when we found that using FIFO yielded a mean of 10.88 robots with a 0.64 standard deviation and similarly, using dSSPT yielded a mean

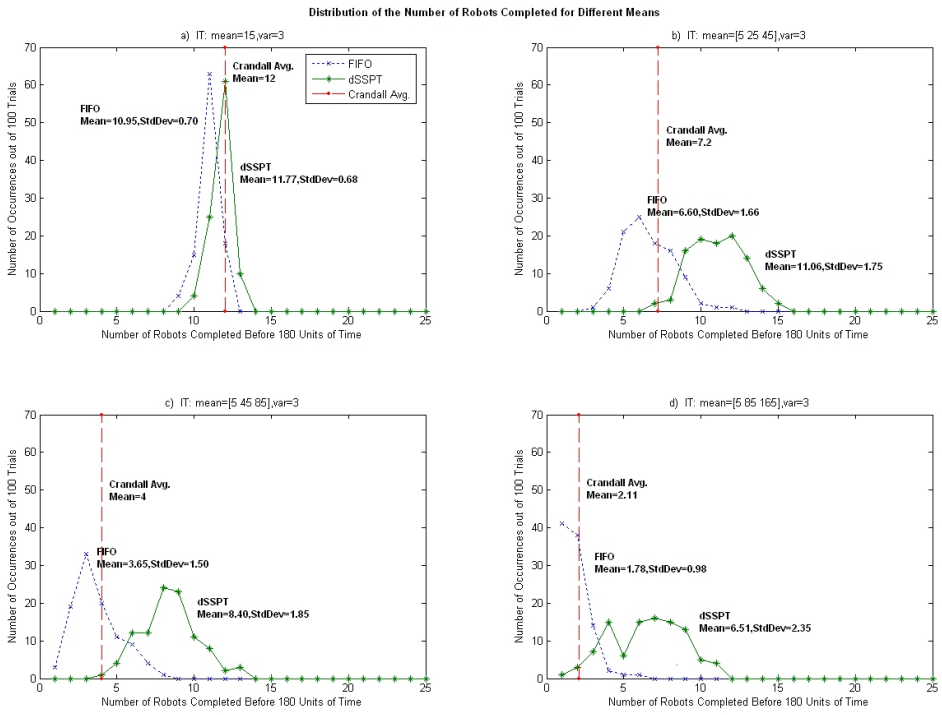


Figure 4.6: Number of robots addressed within NT for a 25 robot team when randomly drawing task ITs.

of 11.55 robots with a 0.63 standard deviation. As the variation increased from 1 to 6, we can see that there was a slight shift to the right in the number of robots that could be addressed within the NT when using dSSPT. As seen in Figure 4.1(b), FIFO yielded a mean IT of 11.08 and standard deviation of 0.71, which is very similar to 8(a), whereas dSSPT's mean increased to 12.24 with a standard deviation of 0.73. This indicates that groups with tasks of varying IT would be more efficient using a dynamic scheduling algorithm as opposed to none or static averages.

This effect is even more pronounced in the experiments with long, medium and short ITs shown in Figure 4.1. Again, this set of experiments varies the mean of the short, medium, and long ITs from 5, 25, and 45 to 5, 85, and 165, respectively, each with a variance of 3. For IT of [5, 25, 45] as shown in Figure 4.1(b), the mean number of robots finishes within NT for FIFO was 6.60 robots whereas dSSPT gave a significantly better mean of 11.06.

Based on fan-out one would expect a mean $NT/IT = 180/25 = 7.2 robots$, which is close to FIFO's mean and significantly less than dSSPT's mean. Similiarly, for IT of [5 45 85] in Fig-

ure 4.1(c), the mean for FIFO is 3.65 and the fan-out average is 4, compared to dSSPT's mean of 8.40. In Figure 4.1(d), where the IT of tasks were [5 85 165], FIFO has a mean of 1.78 with a deviation of 0.98 and the fan-out average is 2.11, whereas dSSPT had a mean of 6.51 and deviation of 2.35. The spread of the dSSPT is much greater since the task sizes were large (with medium mean of 85 and large mean of 165) in comparison to the NT of 180. In these cases, any variety of SPT would tend to schedule the shorter tasks first to minimize downtime. FIFO's sharp peak at 2.06 demonstrates that it will be inefficient in terms of downtime since medium and large tasks will be scheduled as they come, even if it is first in the list.

These experiments show that using an efficient scheduling algorithm to minimize downtime such as dSSPT as opposed to a simplistic one such as FIFO can make a dramatic difference in the number of robots a human operator can address within a given amount of time. From the experiment plots, it can be seen that both FIFO and dSSPT show a nearly Gaussian curve that peaks at the mean. This mean is similar to the average performance of a group. The results of scheduling would yield exactly Olsen et al.'s result for the basic case where all robots have the same IT. However, since in reality the IT will surely vary depending on the task and environment, Goodrich et al.'s static measure of span-of-control does not apply to the general case.

This indeed shows that dSSPT schedules more tasks sooner than FIFO or an average estimate if the human is over-subscribed. However, one parameter was somewhat arbitary: this experiment was only run on teams of 25 robots. Is this a reasonable number of robots on the team? If a user is so over-subscribed, it likely means that s/he is already saturated and is exceeding his/her manageable span-of-control. The next chapter (Chapter 5) looks at the span-of-control (or saturation) problem mathematically and experimentally.

4.2 Multirobot Area Coverage Prospecting Problem

This section describes the experimental platform to help characterize dSSPT through the performance of the multirobot team s/he is controlling. Simulations comparing the two scheduling algorithms for humans (dSSPT and FIFO) examine trends based on varying the parameters for density, density randomness (aperiodicity of tasks), velocity and IT parameters. In each simulation, these human scheduling algorithms are coupled with two possible robot scheduling modes, either with robot task reallocation, or without, thus giving four scheduling results for comparison. The metrics

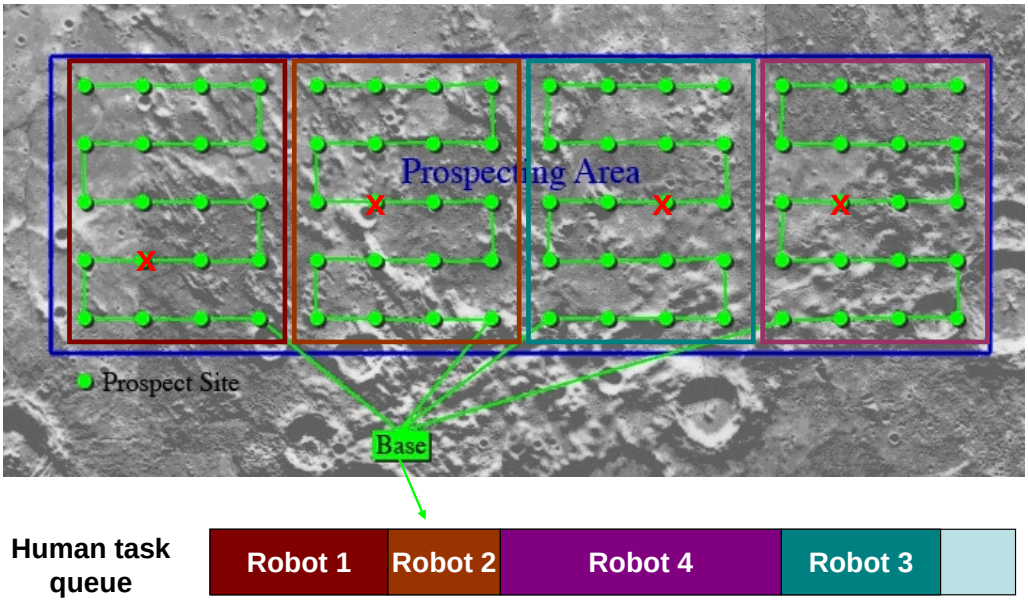


Figure 4.7: An example of a potential area coverage prospecting problem. Each robot has itŠs own area to cover and itŠs own set of tasks to perform (like navigating to waypoints). When the robot encounters an mineral, it stops to alert the base station for a human operator to examine and note it. This mineral becomes a task for the human at the base.

of downtime and robot area coverage are analyzed in this section.

Multirobot Area Coverage Prospecting Supervisory Control Simulator

A simulation of a human-supervised multirobot team prospecting for minerals was developed to test the scheduling algorithms. In it, the team has a mission of exploring an area to look for minerals. When a mineral patch is found, it becomes a task for the human supervisor to note and identify it (Figure 4.2 shows an illustration).

The parameters of this problem include map size (the size of the area that has to be explored), density of mineral occurrence, randomness (or periodicity) of mineral distribution, the interaction time required at each mineral patch found, the velocity of each robot, and the number of robots. Since velocity is the only parameter in this simulation specific to robot performance, homogeneous robots will all have the same velocity and heterogeneous' will differ. Each robot task is assumed to be preemptable (can be interrupted, by human tasks for example, and continued at a later time). Each human task is assumed to be non-preemptable (must be completed all at once). Any minerals found along a robot's path will produce tasks for the human supervisor, which are then scheduled

using a particular algorithm. The results below discuss the use of dSSPT versus FIFO for scheduling human tasks. SPT and SSPT were not included in these experiments since it was previously established that dSSPT outperforms them in terms of downtime, which is related to the other metrics. FIFO is also the most common algorithm used, so stands as a good baseline for comparison.

The robots on the team can operate in two different task allocation modes for their schedules: reallocatable, or non-reallocatable. For both, all robots are initially allocated an area based on their velocity such that in the absence of minerals, they will all finish at the same time. When minerals are found, robots generate human tasks which are scheduled into both the human and the robot's tasklists. In the reallocatable mode, if one robot finishes earlier than other robots, the scheduler reallocates the part of an unexplored area from an unfinished robot to the finished one such that both robots will finish simultaneously if no future minerals are found. The non-reallocatable mode is one in which tasks cannot be shared. This can emulate situations where robots are located far from each other (like Spirit and Opportunity on Mars).

Experiments Varying *IT*

The parameters were varied in a controlled manner to isolate effects. dSSPT and FIFO were compared using both the robot task reallocation mode and without. Each set of simulations was run 30 times. The plots of average values were averaged over these 30 trials and the plots of ratios were taken as an average all trials of ratios within each trial for a fair comparison. 95% confidence intervals were calculated for every data point to determine statistical significance.

Downtime Comparison

In every prospecting experiment, dSSPT had equal or smaller total downtime compared to FIFO. Figure 4.2 plots the comparative downtime for one set of experiments. The table in Figure 4.2 shows the ratios of dSSPT to FIFO downtime for reallocatable and non-reallocatable tasks at various team sizes.

When all human tasks have exactly the same processing time (IT) without variance (the table in Figure 4.2, rows 1, 5-7), dSSPT and FIFO had the same performance with respect to downtime. This makes sense, because if all processing times are constant, then the order will depend only on the release time, according to Figure **??**.

	Parameters					*Span-of-Control*			*Downtime Ratio (dSSPT/FIFO)*				*Area Coverage*	
	Mineral Periodicity	Mineral Density	Robot Velocity	IT ~ $\mathcal{N}$(mean, var)	$\frac{\text{Mean NT}}{\text{Mean IT}}$ +1	FO Upper Bound Estimate **[A]**	FIFO with reallocation **[B]**	dSSPT with reallocation **[C]**	Ratio @ size [A] without reallocation	Ratio @size [A] with reallocation	Ratio @ size [B] with reallocation	Ratio @ size [C] with reallocation	Peak ratio @ size [C] without reallocation	Peak ratio @ size [C] with reallocation
1	Periodic	0.01	1	(15, 0)	6.67+1	8	8	8	1	1	1	1	1	1
2	Periodic	0.01	1	(15, 6)	6.67+1	8	8	8	0.99	0.98	0.98	0.98	1.01	1.01
3	Periodic	0.01	1	([5 15 25], 3)	6.67+1	8	10	10	0.92	0.92	0.84	0.84	1.04	1.07
4	Periodic	0.01	1	([5 25 45], 3)	4+1	5	7	8	0.88	0.88	0.88	0.84	1.06	1.10
5	Random	0.01	1	(15, 0)	6.67+1	8	16	16	1	1	1	1	1	1
6	Random	0.005	1	(15, 0)	13.3+1	14	>20	>20	1	1	1	1	1	1
7	Random	0.02	1	(15, 0)	3.33+1	4	8	8	1	1	1	1	1	1
8	Random	0.01	5	([5 15 25], 3)	1.33+1	2	4	4	0.98	0.98	0.96	0.96	1.07	1.08
9	Random	0.01	[1 5 10]	([5 15 25], 3)	1.25+1	2	6	7	1	1	0.98	0.95	1.04	1.08
10	Random (Size 1)	0.01	1	([5 15 25], 3)	6.67+1	8	12	14	0.91	0.90	0.85	0.81	1.25	1.25
11	Random (Size 1)	0.01	1	([5 25 45], 3)	4+1	5	9	12	0.92	0.93	0.85	0.77	1.17	1.18
12	Clusters Size 5	0.01	1	([5 15 25], 3)	6.67+1	8	19	19	0.96	0.96	0.98	0.98	1.05	1.05
13	Clusters Size 5	0.01	1	([5 25 45], 3)	4+1	5	11	11	0.96	0.95	0.85	0.85	1.05	1.05
14	Clusters Size 20	0.01	1	([5 15 25], 3)	6.67+1	8	20	20	0.98	0.98	0.97	0.97	1.06	1.06
15	Clusters Size 20	0.01	1	([5 25 45], 3)	4+1	5	19	19	1	0.98	0.98	0.98	1.02	1.02

Figure 4.8: Summary of results based on varying parameters listed in the first 4 columns. Row 1 is the baseline, rows 2-4 vary *IT*, rows 5-7 vary mineral density, rows 8 & 9 compare homogenous and heterogeneous robot velocities and rows 10 onward vary mineral clustering distribution.

If the tasks are periodic but the *IT*s vary (the table in Figure 4.2, rows 2-4), then dSSPT shows a marked improvement over FIFO, especially if there are many tasks queued and waiting for human assistance. Two types of variances were considered, Gaussian variance around one mean (row 2) and having Gaussians around several means (row 3-4) to simulate the idea of tasks with varying length—short, medium, long. This is intended to simulate more variegated robot tasks that robots would likely encounter on a real mission. As the variance of *IT* increases, dSSPT performs increasingly better than FIFO, from a 2% improvement in row 2 to 16% in row 4 for robots with reallocation at the dSSPT algorithm saturation size (column 4 of Downtime section).

Human Tasks Over Time

Similar to the uni-resource experiment, it was found that the rate of task completion of over time is much faster with dSSPT compared to FIFO, especially for cases of high variance in *IT*. Figure 4.2

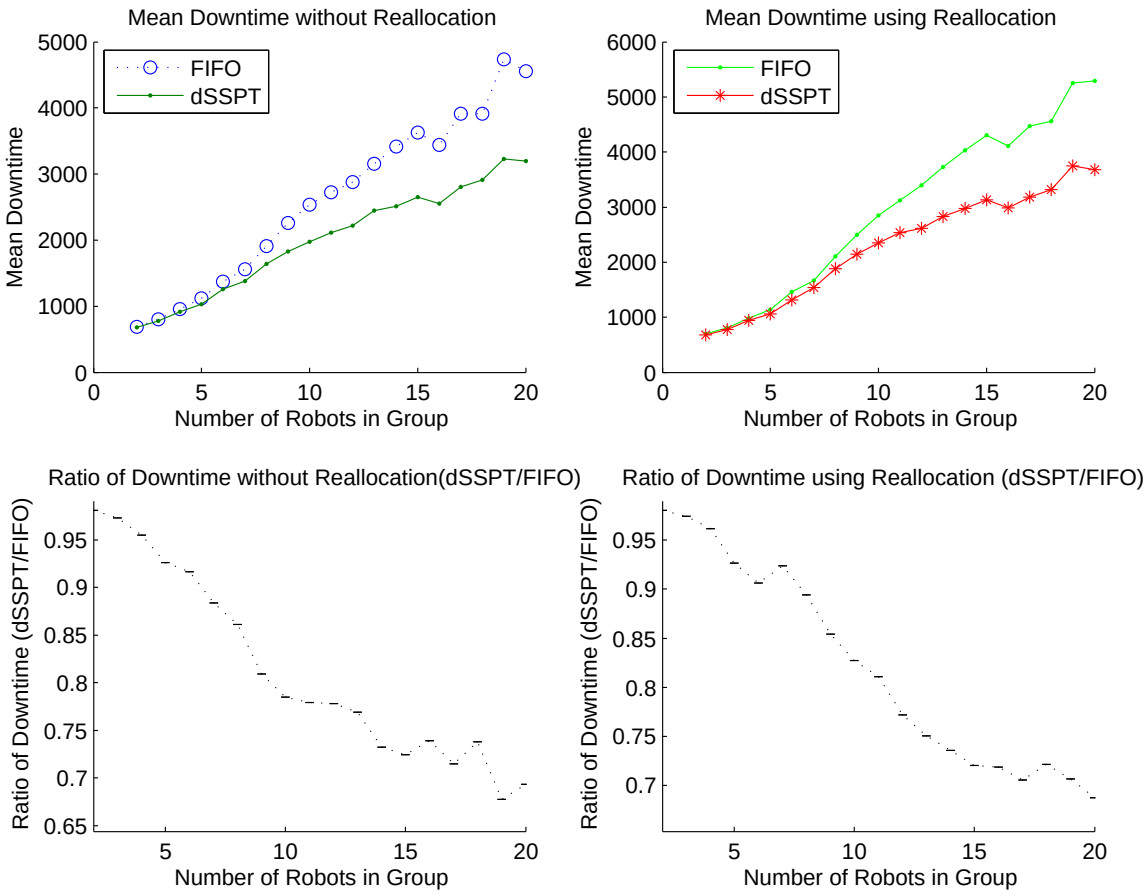


Figure 4.9: Downtime average and ratio compared to FIFO of simulations on 50x50 map, periodic tasks, homogenous velocity of 1m/s, mean IT drawn from normal distributions of mean [5, 25 or 45] and variance 3. The left column shows the experiments without robot task reallocation whereas the right column does have robot task reallocation. The top row illustrates the averages over all trials and the bottom row is the average over all trials of the performance ratio per trial.

shows change in task completion over time for robot teams of varying size (from 2 to 18 robots). It can be seen in the graph of average human task completion over time that FIFO's rate is linearly increasing for all team sizes until it peaks, whereas dSSPT has a higher rate with increasing team sizes. After the peak, the numbers are less significant since when human task completion ends varies depending on the particular simulation. This improved rate of task completion is also reflected in the plot of the ratios of dSSPT over FIFO with ratios nearly always greater than 1 (with 95% confidence), especially from 6 robots onward (though a clear improvement can also be seen with 4 robots). It turns out that for this simulation, saturation occurs from 5 robots upward. In general, the improvement that dSSPT has over FIFO increases as teamsize approaches the expected saturation based on averages (fan-out) and the ratio remains consistently higher from saturation point upwards.

Area Coverage Over Time

dSSPT outperforms FIFO most of the time. Simulations with *IT* with no variance yielded no difference between the two algorithms, as expected. Variance in periodicity of minerals and in *IT* yielded more interesting results. Figure 4.2 shows area coverage over time for the same experiment discussed in the previous section.

For team sizes at the experimental saturation point, dSSPT always outperforms FIFO using scheduling with robot reallocation. With reallocation of remaining robot tasks, the minerals are found sooner since there are fewer idle robots and the supervisor is oversubscribed more often. This allows dSSPT to order the overlapping tasks more efficiently to reduce downtime. Without reallocation, FIFO sometimes outperforms dSSPT towards the end of the mission because dSSPT is scheduling many of the smaller human tasks earlier on, thus freeing the robots to cover more area in the beginning, but leaving the longer human tasks until later, so the robots waiting on the human towards the end cover less area. Since their tasks are not reallocatable to other robots, they must wait. This problem is less prominent in scheduling with reallocation because in that case, when the robot is busy waiting for the human, other free robots can take over its remaining coverage area. The extreme oversubscription when the team is larger than its saturation size demonstrates the drawback of SPT-based heuristic—longer tasks may be pushed back.

An example of this is seen in Figure 4.2, where FIFO and dSSPT with reallocation are compared. Team sizes smaller than the saturation point tend to have larger variances in area coverage

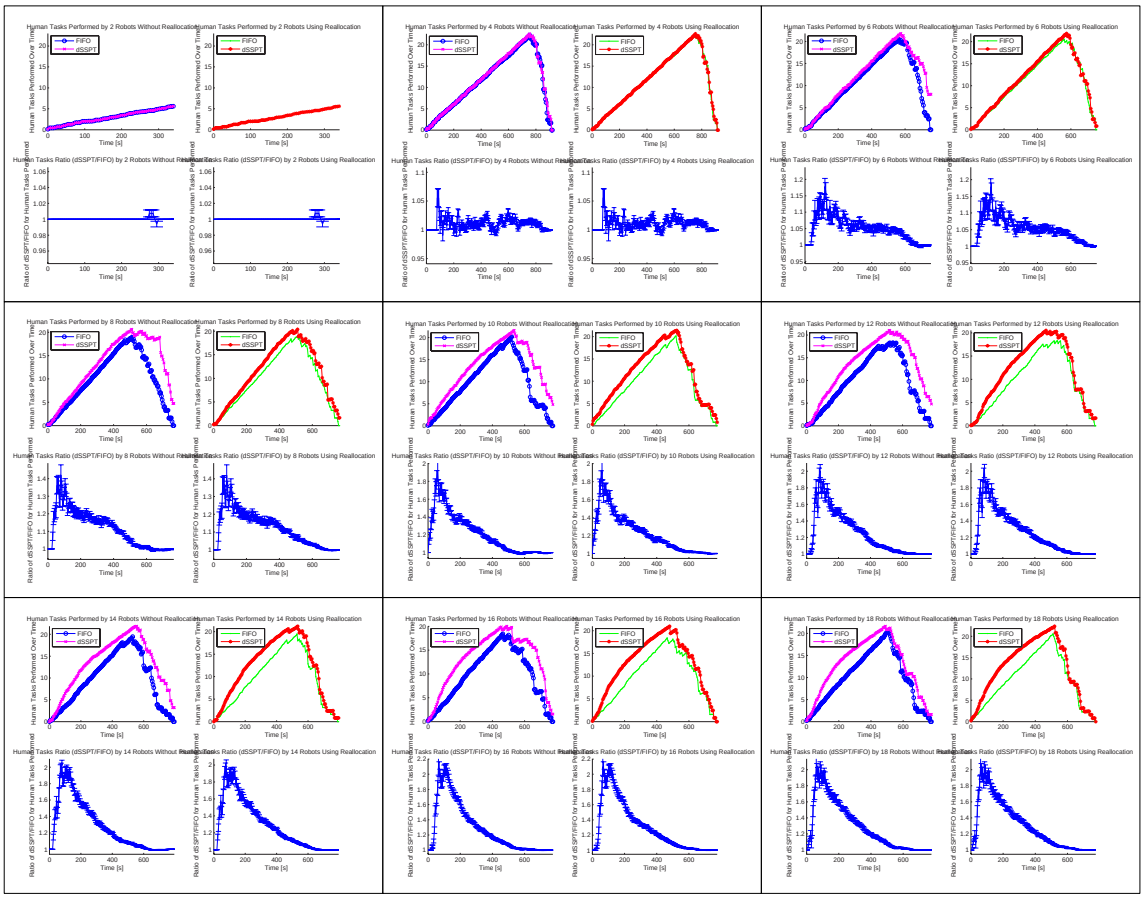


Figure 4.10: Human task completion over time average and ratio compared to FIFO of simulations on 50x50 map, periodic tasks, homogenous velocity of 1m/s, mean IT drawn from normal distributions of mean [5, 25 or 45] and variance 3. Subplots are of increasing team size from 2 to 18, left to right, top to bottom. The left column of each subplot shows the experiments without robot task reallocation whereas the right column does have robot task reallocation. The top row illustrates the averages over all trials and the bottom row is the average over all trials of the performance ratio per trial.

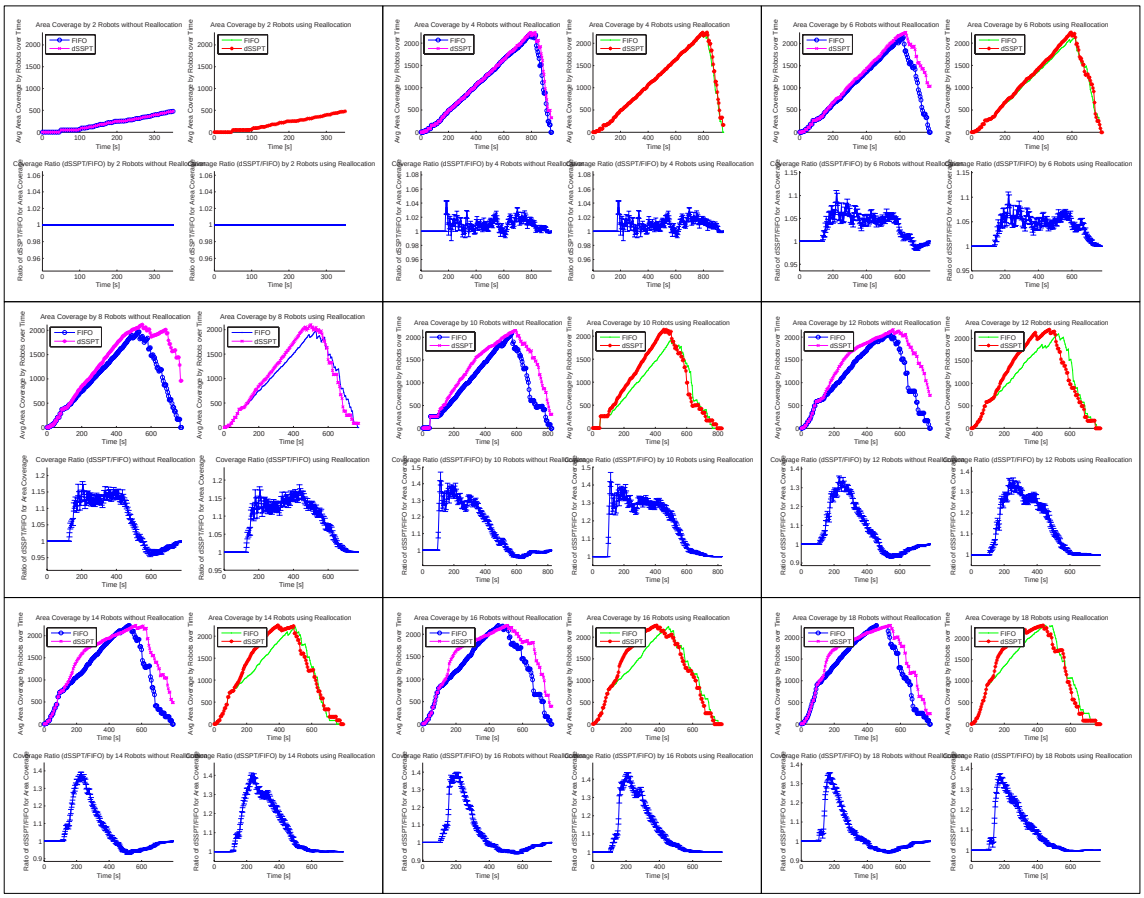


Figure 4.11: Robot area coverage over time average and ratio compared to FIFO of simulations on 50x50 map, periodic tasks, homogenous velocity of 1m/s, mean IT drawn from normal distributions of mean [5, 25 or 45] and variance 3. Subplots are of increasing team size from 2 to 18, left to right, top to bottom. The left column of each subplot shows the experiments without robot task reallocation whereas the right column does have robot task reallocation. The top row illustrates the averages over all trials and the bottom row is the average over all trials of the performance ratio per trial.

performance. When team sizes approach saturation, dSSPT again outperforms FIFO. For team sizes larger than saturation point (5 in this case), as team size increases, dSSPT's peak performance over FIFO progressively shifts earlier. For a team size of 5 onwards dSSPT always outperforms FIFO in terms of area coverage with reallocation (the ratio between the performances is seen in the lower graph to be greater than 1). Without reallocation, we can see the little dip below a ratio of one towards the end of the mission. This dip pushes forward as team size increases.

Experiments Varying *NT*

The parameters were varied in a controlled manner to isolate effects. Parameters affecting *NT* include density, randomness of mineral occurance and velocity of robots. As above, dSSPT and FIFO were compared using both the robot task reallocation mode and without. Each set of simulations was run 30 times. The plots of average values were averaged over these 30 trials and the plots of ratios were taken as an average all trials of ratios within each trial for a fair comparison. 95% confidence intervals were calculated for every data point to determine statistical significance.

Downtime Comparison

The table in Figure 4.2 rows 5-7 show that despite a change in density, as long as *IT* variance is zero there is no effect on downtime. Rows 8-9 show that with changing velocity (homogeneous team versus heterogeneous team) resulting in larger variance in *NT*, downtime is not significantly affected (although span-of-control is).

The randomness of mineral occurance was tested as show in the table in Figure 4.2 rows 10-15. Trials were run with mineral occurance being totally random (with each mineral of found in a $1m^2$ distributed randomly), minerals occuring in $5m^2$ clusters distributed randomly, minerals occuring in $20m^2$ clusters distributed randomly, and periodic. An example of this clustering is illustrated in Figure 4.2

As the amount of clustering increases, it acts like the map is becoming more sparse. The improvement in downtime of dSSPT over FIFO decreases as seen in the table in Figure 4.2. It seems like with clustering, even though the global density of area stays the same, the local densities vary with pockets of dense patches and other areas very sparse. When the dense patches occur and many mineral tasks arrives for the human, scheduling with dSSPT does improve downtime over FIFO.

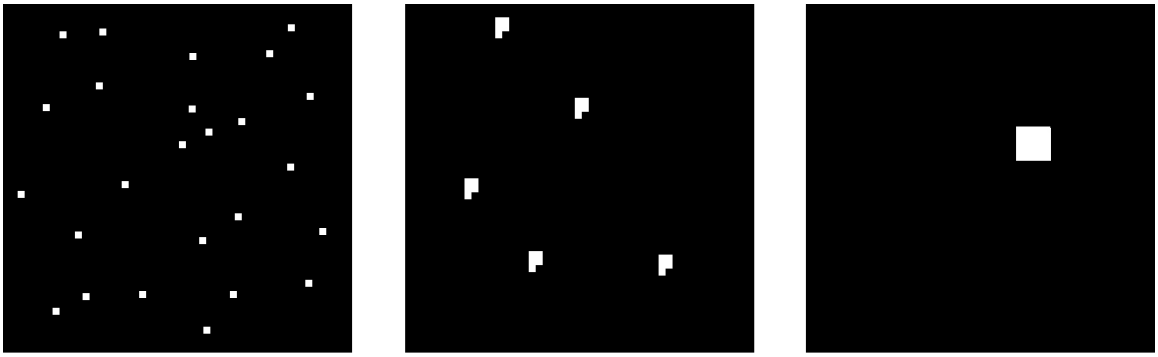


Figure 4.12: Examples to illustrate from left to right, randomly distributed clusters of size $1m^2$, $5m^2$ and $20m^2$

But for the sparse patches, nothing needs to be scheduled so there is no difference between dSSPT and FIFO. A good experiment to run for the future would be to check what density at randomly $1m^2$ cluster size corresponds to another density at a $5m^2$ cluster size.

Area Coverage Over Time

Area coverage is greatly dependent on the downtime experienced by the team. This correlation can be seen throughout rows 5 to 15 of the table in Figure 4.2. Thus, similar to the findings for downtime, it is found that dSSPT has no improvement over FIFO when IT variance is zero, no significant improvement for change in velocity, and increasing cluster size led to less improvent of dSSPT since it acted like a sparser area.

Summary of Results

- Downtime occurs with variance in IT
- Randomness and density of distribution affects downtime and is insufficiently modelled by an average NT and IT
- dSSPT outperforms FIFO, SPT and SSPT in terms of lower downtime
- dSSPT allows more human tasks completed sooner compared to FIFO
- dSSPT usually outperforms FIFO in terms of faster area coverage

dSSPT outperforms FIFO in terms of lower downtime for aperiodic tasks (e.g., random mineral distribution) of varying IT. This results in a faster rate of area coverage for the simulated prospecting problem; in other words, more human tasks are completed sooner thus freeing the robots to perform its tasks sooner.

These results agree with the previous uni-resource experiment in that dSSPT prioritized tasks for the human supervisor such that more human tasks were addressed sooner and allowed the multirobot team to generally covered more area in less time.

Distributions with the same global NT and IT when clustered (or less random) result in performances that reflect a sparser distribution of single minerals. It seems that given the results of the clustering experiments, the averaging implied with using NT and IT for any problem is insufficient to model the entire problem. Perhaps an entropy parameter is necessary.

Chapter 5

Maximum Team Size Simulations

5.1 Saturation in Multirobot Area Coverage Prospecting Problem

This section analyzes the multirobot area coverage prospecting problem to find team saturation both mathematically and experimentally.

Relating Multirobot Area Coverage Prospecting to HMRI Metrics

In relating the parameters of this simulated problem to the HMRI metrics, NT can be expressed as:

$$\begin{aligned} NT &= \frac{\textit{Time to cover area given no minerals in the area}}{\textit{Expected number of minerals}} \\ &= \frac{\frac{area}{velocity}}{area \times density} \\ \therefore NT &= \frac{1}{velocity \times density} \end{aligned} \tag{5.1}$$

Saturation occurs when the human cannot address any more robot tasks (the robot team is large enough that there is not enough idle time left to address more tasks). At saturation, the completion time of the team is limited by the human's task completion time. In this case, increasing team size would not lower the mission completion time, since the human cannot work any faster than one task at a time.

The minimum time to complete the mission for the prospecting problem is:

$$\begin{aligned} \textit{Minium Mission Completion Time} &= \frac{\textit{Time to cover area sans minerals}}{\textit{Number of robots}} \\ &+ \frac{\textit{Number of minerals} \times \textit{Time for each mineral task}}{\textit{Number of robots}} \\ &= \frac{\frac{area}{velocity} + area \times density \times IT}{\textit{Number of robots}} \end{aligned} \tag{5.2}$$

The curve of this function can be seen as the dotted red line in Figure 5.1b. Note that this is the minimum completion time because the equation does not consider the number or length of mineral tasks encountered by each robot which may not be evenly distributed.

The completion time for human saturation (or minimum human completion time—a human cannot possibly complete the tasks any faster) is equal to the amount of time it takes to address mineral tasks without consideration of human idle time:

$$\textit{Human Saturation} = area \times density \times IT \tag{5.3}$$

This threshold for human completion time is the dashed horizontal red line in Figure 5.1b. This equation is the minimum human completion time, in that it does not factor in any human idle time at all.

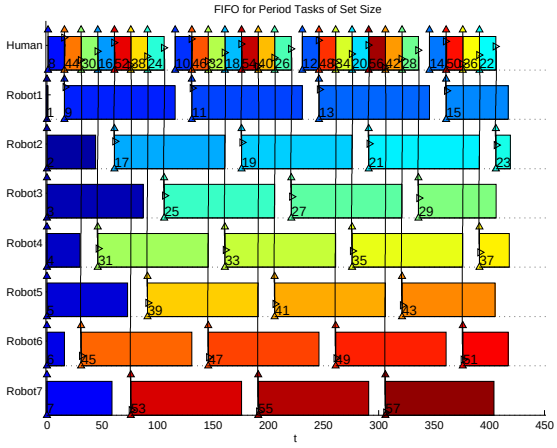
The point where these two curves intersect is the minimum saturation point where the mission completion time is equal to the human completion time. Equating Equations **??** and 5.3 and solving for the number of robots:

$$\begin{aligned} \frac{\frac{area}{velocity} + area \times density \times IT}{\textit{Number of robots}} &= area \times density \times IT \\ \therefore \textit{Number of robots} &= \frac{1}{area \times density \times IT} + 1 \end{aligned} \tag{5.4}$$

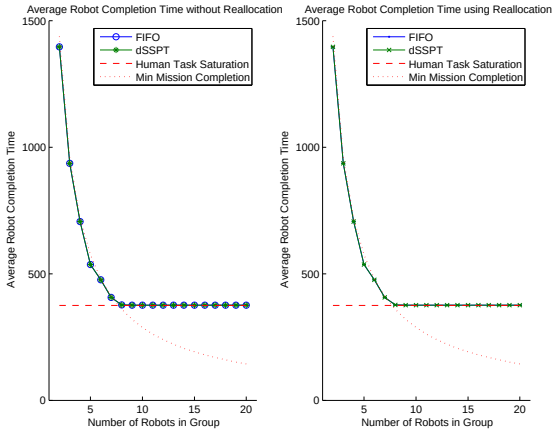
Substituting NT from Equation 5.1 into Equation 5.4 we get:

$$\textit{Number of robots} = \frac{NT}{IT} + 1$$

This is exactly the same as the upper bound for fan-out in Equation 2.1. This demonstrates that only under the ideal conditions of no variance in NT (i.e., periodicity) nor IT will the FO prediction for saturation occur.



(a) Left: Timeline of task



(b) Right: Average human completion time of trials compared to expected human saturation time (dashed) and expected mission completion time (dotted)

Figure 5.1: Plots from an experiment with no variance in IT, periodic tasks, and homogenous robots.

Baseline: No Variance, Periodic Tasks

As a baseline to demonstrate the situation of ideal saturation implied by the FO (Equation 2.1), an experiment was run with periodic occurrence of minerals (no randomness), same velocity of 1m/s for all robots, a fixed IT of 15s with no variance and a density of 0.01 (NT = 100s) in a $50 \times 50 m^2$ map. The results can be seen in Figure 5.1.

Figure 5.1a shows an example of what a simulation with 7 robots looks like on a timeline plot showing both robot and human tasks.

Saturation

The most interesting finding to note is that the saturation point exactly corresponds to the predicted FO. Based on Equation 2.1, the expected saturation point would be $\frac{100}{15} + 1 = 7\frac{2}{3}$ robots. Figure 5.1b shows that human saturation does indeed occur before a team size of 8 and remains saturated beyond that size. Teams smaller than 8 result in a human completion time close to and slightly less than the expected minimum completion time. Figure 5.1a shows that at a team size of 7, the human still has some idle time and is not yet completely saturated. This is reflected in Figure 5.1b where the completion time is still above the saturation point.

Same performance

Since these periodic tasks had the same IT, dSSPT (like SPT) would not reorder the human tasks differently than FIFO order. Also, since the tasks were evenly spaced throughout the area all robots finished at similar times and no robot task reallocation occurred. NT/IT also remained constant over time for a team size of 7 or lower since the human is under-subscribed.

Non-zero variance, Aperiodic Tasks: Human Saturation

This section compares theoretical span-of-control (FO) to actual span-of-control for simulations. For a fair comparison, only the simulations with robot reallocation are discussed because this mode dynamically redistributes remaining tasks amongst available robots to lower robot completion times. It also reallocates area such that the human tasks generally appear sooner, thus reducing human idle time (which is what we want to do in order to reach the saturation point). The caveat is that the reallocation mode is an online heuristic algorithm and not guaranteed to minimize completion time. A better robot task reallocation algorithm can potentially decrease the saturation point. But the relative difference in saturation between FIFO and dSSPT should remain the same.

Section 5.1 above showed that when tasks occur periodically and all with the same IT, the saturation point (which defines the span-of-control) does indeed coincide with the theoretical fan-out. This section shows that when variances occur in IT or NT (i.e., the tasks are released at random times), the saturation point increasingly drifts away from the FO estimate.

Human completion time

When the tasks are periodic but the ITs vary (table in Figure 4.2, rows 2-4), if the variance is small the saturation is essentially the same as theoretical FO. However, when IT variance is increased, the saturation point gets pushed to larger team sizes. In row 3 with mean IT of 5, 15 or 25s (the mean is randomly selected from the three values) and variance 3, both FIFO and dSSPT's saturation occurs at a team size of 10 instead of the predicted 8. As IT varies even more as in row 4 with mean IT of 5, 25 or 45s, not only does the saturation occur after the FO estimate of 5, but dSSPT starts to have a slightly larger saturation compared to FIFO (8 compared to 7) as can be seen in Figure 5.1(a).

For the cluster experiments shown in Figure 4.2, rows 10-15, the saturation point can be seen to increase as the cluster size increases (comparing rows 10, 12, 14 and rows 11, 13, 15). This again

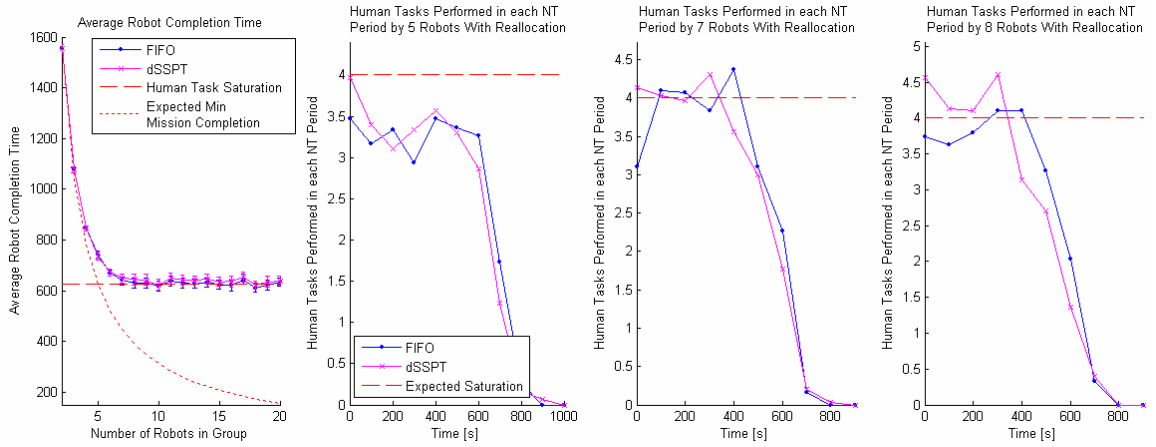


Figure 5.2: Experiment with periodic tasks, homogenous velocity of 1m/s, mean IT of 5, 25 or 45. (a) Left: Human completion time compared to expected human saturation and mission completion. (b-d) Right: Number of tasks addressed in each NT period for different team sizes.

indicates that with larger cluster size and lower cluster density, the effect is the as lowering mineral density in the area for random $1m^2$ minerals.

The reason why the saturation occurs later is due to the varying nature of the tasks (in terms of periodicity and size) as well as the finite timeline of the mission, both of which would be expected in any real application. Figure 5.1(b) illustrates that when tasks occur aperiodically with the expected average NT, even when they have the same IT there are instances where the human has idle time, as well as periods where they are over-subscribed. It is in those periods of idle time where potentially more robot tasks can be allotted, thus increasing the span-of-control. Similarly, Figure 5.1(c) shows that when tasks occur periodically but have varying ITs, there are again periods of idle time and over-subscription for the human. When dSSPT reorders non-preemptable tasks, if it reorders the first task in the list, idle time gets inserted into the schedule. This difference occasionally pushes dSSPT to have a slightly larger span-of-control than FIFO.

Therefore, in order for the human to be fully saturated, there must be no idle time at all (or at least not enough to fit an average task). More robots mean that more tasks are released earlier for the human, which reduces idle time.

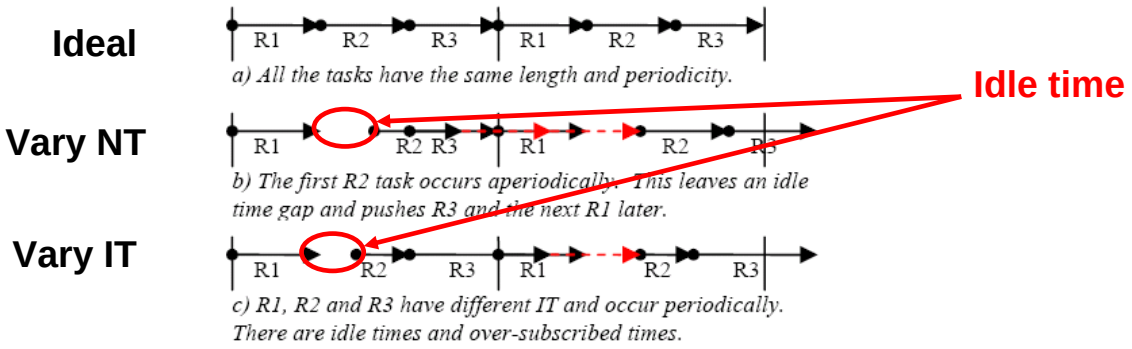


Figure 5.3: Examples of how variance in NT and IT incur idle times and over-subscribed times. This allows more tasks to "fit" in.

NT/IT Over Time

Since the estimated saturation, NT/IT, is just an average, the actual number of tasks addressed in any NT period may differ. For example, when there are periodic tasks with mineral density of 0.01 and IT of 5, 25 or 45s with variance 3, the expected FO is 5. However, the previous section demonstrated that the actual saturation did not occur until 7 for FIFO and 8 for dSSPT, as shown in the human completion time graph of Figure 5.1(a). Taking a look at this example over intervals of time (Figure 5.1(b-d)) we can see that indeed at a team size of 5, the expected number of tasks addressed in that time frame (NT/IT) is 4, but the actual number of tasks addressed is much lower at around 3.3 during "steady state" (excluding three points of the tail where the mission is winding up). However, at a team size of 7, we can see that the mean for FIFO is indeed close to 4 at 3.8, and at a team size of 8, dSSPT also has NT/IT at 3.9. This again reiterates that the saturation size for a team operating in a dynamic world is higher than the average-based estimate for FO.

Figure 5.1(b-d) shows that the number of tasks addressed over time also varies depending on the algorithm. dSSPT schedules more tasks earlier on, so it has a shorter steady-state period compared to FIFO, with a higher NT/IT in the beginning and an earlier decrease to zero. This suggests that it might be more efficient to have dynamic team sizes depending on the scheduling algorithm used. For dSSPT, it seems that a dynamic team size with a larger team at the beginning and a smaller one as NT/IT decreases would be most efficient. Those robots that are unneeded for latter portions of the mission can potentially move on to start other missions sooner.

Summary of Results

- General factors that affect maximum team size are:
  - Variance in NT (e.g., velocity, randomness or clustering)
  - Variance in IT (e.g., tasks of different lengths)
  - Scheduling algorithm
- Actual maximum team size will be larger than Fan-Out prediction for any real situation
- Monitoring tasks performed in each NT period over time is a good gauge of whether a human supervisor is saturated. It is also potentially a good method for doing dynamic team sizing

When there is any variance in either NT or IT, the actual span-of-control becomes larger than the average-based prediction of FO. This means that the upper bound of the idealized FO equation is actually going to be a lower bound on the maximum team size for any practical situation.

Analysis of these scheduling algorithms for span-of-control show that actual span-of-control is greater than the average-based prediction given by fan-out equation's upper bound. Additionally, analysis of NT/IT over time showed that the number of tasks performed in each NT period dropped off towards the latter part of the mission, with the drop-off occurring sooner for dSSPT than FIFO. This trend suggests that dynamic team sizing would the make most use of the robot resources and also that span-of-control is affected by the scheduling algorithm used.

Chapter 6

Conclusions

As supervisory control of multirobot teams become more commonly used, there is an increasing need for efficient utilization of the human supervisor's time to maximize team performance. This research suggests one method of efficiently using human time is to prioritize human tasks such that robots regain autonomy sooner. Due to the nature of human tasks as precedence constraints for robots, downtime minimization through the dSSPT algorithm results in higher team efficiency. Based on a simulated prospecting mission, dSSPT managed to prioritize tasks for humans such that the team generally covered more area in less time.

Analysis of these scheduling algorithms for span-of-control shows that actual span-of-control is greater than the average-based prediction given by fan-out equation's upper bound. The factors which affect span-of-control include variance in NT (from heterogeneity and randomness or clustering of tasks), variance in IT and the particular scheduling algorithm used. Additionally, analysis of NT/IT over time showed that the number of tasks performed in each NT period dropped off towards the latter part of the mission, with the drop-off occurring sooner for dSSPT than FIFO. This trend suggests that dynamic team sizing would the make most use of the robot resources and also that span-of-control is affected by the scheduling algorithm used.

6.1 Applications of this Research in General

The facts learned from this body of research can be summarized as general lessons for human-multirobot interaction:

- If you have a better scheduling algorithm, you can better utilize human time
- Maximum team size estimates based on averages are too conservative
- Most importantly, we should be aware of how the use of human time (in particular with scheduling) effects team performance in multirobot supervisory control and formalize the way we look at it

6.2 Future Work

Potential future work includes dynamic team-sizing experiments, and use of a centralized scheduler for both the human and the multirobot team as opposed to this decentralized scheduling (where the human tasks are scheduled without knowledge of the robots' tasklists). The potential gain from using centralized scheduling would be that more information might provide a better schedule. For example, if the robots had inter-robot dependences (such as precedence constraints on tasks between robots), the downtime for robots can potentially be further minimized if the robot tasks are reordered as well. Another possible direction would be to develop a scheduling algorithm for more diverse types of tasks which the human might encounter, including both preemptive and non-preemptive tasks.

6.3 Acknowledgments

My sincerist thanks to my advisor John for his support, guidance and encouragement. I would also like to thank the people on my committee, Kristen and Illah, for listening to my ideas and providing feedback and the people from T-SAR Lab and Prospect Team for providing the inspiration and advice. I also appreciate the love and support from my family, friends and Rob. And let me not forget those at the post office.

This work was supported in part by NASA under Cooperative Agreement No. NNA05CP96A.

References

[1] Mars exploration rover. Online, 12 Jun 2006.

[2] Mars exploration rover mission: People. Online, 2007.

[3] J. Adams. Supporting human supervision of multiple robots. *The Journal of the Robotics Society of Japan*, 24(5):17–19, July 2006.

[4] N. Bansal. Algorithms for flow time scheduling, 2003.

[5] D. P. Bunde. Spt is optimally competitive for uniprocessor flow. *Inf. Process. Lett.*, 90(5):233–238, 2004.

[6] J. W. Crandall and M. L. Cummings. Developing performance metrics for the supervisory control of multiple robots. In *HRI '07: Proceeding of the ACM/IEEE international conference on Human-robot interaction*, pages 33–40, New York, NY, USA, 2007. ACM Press.

[7] M. L. Cummings and P. M. Mitchell. Management of multiple dynamic human supervisory control tasks for UAVs. Human Computer Interaction International Human Systems Integration Conference, 2005.

[8] A. Elfes, J. Dolan, G. Podnar, S. Mau, and M. Bergerman. Safe and efficient robotic space exploration with tele-supervised autonomous robots. In *Proceedings of the AAAI Spring Symposium*, pages 104 – 113., March 2006. to appear.

[9] W. Ferrell and T. Sheridan. Supervisory control of remote manipulation. *IEEE Spectrum*, 4(10), 1967.

[10] M. A. Goodrich and E. R. Boer. Model-based human-centered task automation: A case study in acc design. In *IEEE Transactions on Systems, Man, and Cybernetics — Part A: Systems and Humans*, volume 33, pages 325–336, May 2003.

[11] S. Hua and G. Qu. A new quality of service metric for hard/soft real-time applications. In *ITCC '03: Proceedings of the International Conference on Information Technology: Computers and Communications*, page 347, Washington, DC, USA, 2003. IEEE Computer Society.

[12] B. L. W. MacCarthy and S. J. R. Crawford. Human performance in industrial scheduling: A framework for understanding. *HUMAN FACTORS AND ERGONOMICS IN MANUFACTURING*, Volume 11, Issue 4:299–320, 2001.

[13] D. Olsen and M. Goodrich. Metrics for evaluating human-robot interactions. In *NIST Performance Metrics for Intelligent Systems*, Sep 2003.

[14] J. A. Stankovic, M. Spuri, M. D. Natale, and G. C. Buttazzo. Implications of classical scheduling results for real-time systems. *Computer*, 28(6):16–25, 1995.

[15] A. Steinfeld, T. Fong, D. Kaber, M. Lewis, J. Scholtz, A. Schultz, and M. Goodrich. Common metrics for human-robot interaction. In *HRI '06: Proceeding of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 33–40, New York, NY, USA, 2006. ACM Press.

[16] E. Svodoba. Astronomers planning close-ups of mars from (of all things) a balloon. Online, 13 Jun 2006.

[17] S. Wang. One (martian) year and counting. Online, 12 Jan 2006.